

SN74ACT8800 Family

**32-Bit CMOS Processor
Building Blocks**

Data Manual

1990

| | |
|---|-----------|
| General Information | 1 |
| SN74ACT8818A 16-Bit Microsequencer | 2 |
| SN74ACT8832A 32-Bit Registered ALU | 3 |
| SN74ACT8836A 32- × 32-Bit Parallel Multiplier | 4 |
| SN74ACT8841A Digital Crossbar Switch | 5 |
| SN74ACT8847 64-Bit Floating-Point/Integer Unit | 6 |
| SN74ACT8847 Application Information | 7 |
| SN74ACT8867 32-Bit Vector Processor Unit | 8 |
| Design Support | 9 |
| Mechanical Data | 10 |

SN74ACT8800 Family 32-Bit CMOS Processor Building Blocks

Data Manual

Copyright © 1990, Texas Instruments Incorporated

First edition: March 1988

First revision: June 1988

Second revision: June 1989

Third revision: October 1990



IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to or to discontinue any semiconductor product or service identified in this publication without notice. TI advises its customers to obtain the latest version of the relevant information to verify, before placing orders, that the information being relied upon is current.

TI warrants performance of its semiconductor products to current specifications in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Unless mandated by government requirements, specific testing of all parameters of each device is not necessarily performed.

TI assumes no liability for TI applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Texas Instruments products are not intended for use in life-support appliances, devices or systems. Use of a TI product in such applications without the written consent of the appropriate TI officer is prohibited.

Copyright © 1990, Texas Instruments Incorporated

First edition: March 1988

First revision: June 1988

Second revision: June 1989

Third revision: October 1990



INTRODUCTION

In this manual, Texas Instruments presents technical information on the TI SN74ACT8800 family of 32-bit processor "building block" circuits. The SN74ACT8800 family is composed of single-chip VLSI processor functions, all of which are designed for high-complexity processing applications.

This manual includes specifications and operational information on the following high-performance advanced-CMOS devices:

- SN74ACT8818A 16-bit microsequencer
- SN74ACT8832A 32-bit registered ALU
- SN74ACT8836A 32- × 32-bit parallel multiplier/accumulator
- SN74ACT8841A Digital crossbar switch
- SN74ACT8847 64-bit floating-point/integer unit
- SN74ACT8867 32-bit vector processor unit

These high-speed devices operate at or above 20 MHz, while providing the low power consumption of TI's advanced 1- μ m EPIC™ CMOS technology. The EPIC™ CMOS process combines twin-well structures for increased density with 1- μ m gate lengths for increased speed.

The *SN74ACT8800 Family Data Manual* contains design and specification data for all five devices previously listed and includes additional programming and operational information for the 'ACT8818A, 'ACT8832A, and 'ACT8847. Two application notes, "Chebyshev Routines for the SN74ACT8847" and "High-speed Vector Math and 3-D Graphics Using the SN74ACT8847 Floating Point Unit" are also included.

Introductory sections of the manual include an overview of the '8800 family and a summary of the software tools and design support TI offers for the chip-set. The general information section includes an explanation of the function tables, parameter measurement information, and typical characteristics related to the products listed in this volume.

Package dimensions are given in the Mechanical Data section of the book in metric measurement (and parenthetically in inches).

Complete technical data for any Texas Instruments semiconductor product is available from your nearest TI field sales office, local authorized TI distributor, or by calling Texas Instruments at 1-800-232-3200.

EPIC is a trademark of Texas Instruments Incorporated.

INTRODUCTION

In this manual, Texas Instruments presents technical information on the TI SN74ACT8800 family of 32-bit processor "building block" circuits. The SN74ACT8800 family is composed of single-chip VLSI processor functions, all of which are designed for high-complexity processing applications.

This manual includes specifications and optional information on the following high-performance advanced CMOS devices:

- SN74ACT8878A 18-bit microprocessor
- SN74ACT8832A 32-bit registered ALU
- SN74ACT8838A 32- x 32-bit parallel multiplier/accumulator
- SN74ACT8841A Digital crossover switch
- SN74ACT8847 64-bit floating-point integer unit
- SN74ACT8887 32-bit vector processor unit

These high-speed devices operate at or above 20 MHz, while providing the low power consumption of TI's advanced 1-µm EPIC™ CMOS technology. The EPIC™ CMOS process combines twin-well structures for increased density with 1-µm gate lengths for increased speed.

The SN74ACT8800 Family Data Manual contains design and specification data for all five devices previously listed and includes additional programming and operational information for the ACT8818A, ACT8832A, and ACT8847. Two application notes, "Chebyshev Routines for the SN74ACT8847" and "High-speed Vector Math and 3-D Graphics Using the SN74ACT8847 Floating Point Unit," are also included.

Introductory sections of the manual include an overview of the 8800 family and a summary of the software tools and design support TI offers for the chip-set. The general information section includes an explanation of the function tables, parameter measurement information, and typical characteristics related to the products listed in this volume.

Package dimensions are given in the Mechanical Data section of the book in metric measurement (and parenthetically in inches).

Complete technical data for any Texas Instruments semiconductor product is available from your nearest TI field sales office, local authorized TI distributor, or by calling Texas Instruments at 1-800-232-3200.

EPIC is a trademark of Texas Instruments Incorporated.

| | |
|---|-----------|
| General Information | 1 |
| SN74ACT8818A 16-Bit Microsequencer | 2 |
| SN74ACT8832A 32-Bit Registered ALU | 3 |
| SN74ACT8836A 32- × 32-Bit Parallel Multiplier | 4 |
| SN74ACT8841A Digital Crossbar Switch | 5 |
| SN74ACT8847 64-Bit Floating-Point/Integer Unit | 6 |
| SN74ACT8847 Application Information | 7 |
| SN74ACT8867 32-Bit Vector Processor Unit | 8 |
| Design Support | 9 |
| Mechanical Data | 10 |

| | |
|----|---|
| 1 | General Information |
| 2 | ENVIA-150-25 15-25 Microprocessor |
| 3 | ENVIA-150-25 15-25 Integrated ALU |
| 4 | ENVIA-150-25 15-25 Parallel Multiplier |
| 5 | ENVIA-150-25 15-25 Digital Counter/Divider |
| 6 | ENVIA-150-25 15-25 Binary-Decimal Converter |
| 7 | ENVIA-150-25 15-25 Arithmetic Information |
| 8 | ENVIA-150-25 15-25 Vector Processor Unit |
| 9 | ENVIA-150-25 15-25 Floating Point Unit |
| 10 | Mechanical Data |

General Information

Introduction

Texas Instruments SN74ACT8800 family of 32-bit processor building blocks has been developed to allow the easy, custom design of functionally sophisticated, high-performance processor systems. The '8800 family is composed of single-chip, VLSI devices, each of which represents an element of a CPU.

The '8800 chip set provides the performance, functionality, and flexibility to fill the most demanding processing needs and is structured to reduce system design cost and effort. Most of these high-speed processor functions operate at 25 MHz and above, and, at the same time, provide the power savings of TI's advanced, 1 μm EPIC™ (Enhanced Performance Implanted CMOS) CMOS technology.

Geared for computationally intensive applications, SN74ACT8800 devices include high-performance ALUs, multipliers, microsequencers, and both vector and floating-point processor units.

The family's building block approach allows the easy, "pick-and-choose" creation of customized processor systems, while the devices' high level of integration provides cost-effectiveness.

Designed especially for high-complexity processing, the devices in the '8800 family offer a range of functional options. Device features include three-port architecture, double-precision accuracy, optional pipelined operation, and built-in fault tolerance.

Array, digital signal, image, and graphics processing can be optimized with '8800 devices. Other applications are found in supermini and fault-tolerant computers, and I/O and network controllers.

TI's '8800 32-bit processor building block family comprises the following functions:

- SN74ACT8818A 16-bit microsequencer
- SN74ACT8832A 32-bit registered ALU
- SN74ACT8836A 32- \times 32-bit parallel multiplier/accumulator
- SN74ACT8841A digital crossbar switch
- SN74ACT8847 64-bit floating-point and integer unit
- SN74ACT8867 32-bit vector processor unit
- Bipolar Support Chips
- SN74AS8838 32-bit barrel shifter

EPIC is a trademark of Texas Instruments Incorporated.

20 MIPS and Low CMOS Power Consumption

With instruction cycle times of 40 ns or less and the low power consumption of EPIC™ CMOS, the '8800 chip set offers an unrivaled speed/power combination. Unlike traditional microprocessors, which require multiple cycles to perform an operation, the 'ACT8800 processors typically can complete instructions in a single cycle.

The 'ACT8832A registered ALU and 'ACT8818A microsequencer together create a powerful 25-MHz CPU. Because instructions can be performed in a single cycle, the 'ACT8832A/'ACT8818A combination is capable of executing 25 million instructions per second (MIPS).

For math-intensive applications, the 'ACT8836A fixed-point multiplier/accumulator (MAC), 'ACT8847 64-bit floating-point and integer unit and 'ACT8867 32-bit vector processor unit offer unprecedented computational power.

The exceptional performance of the '8800 family is made possible by TI's EPIC™ CMOS technology. The EPIC™ CMOS process combines twin-well structures for increased density with 1- μ m gate lengths for increased speed.

Customized Solution

The '8800 family is designed with a variety of architectural and functional options to provide maximum design flexibility. These device features allow the creation of "customized" solutions with the '8800 chipset.

A **building block approach** to processing allows designers to match specialized hardware to their specific design needs. The 'ACT8818A/'ACT8832A combination forms the basis of the system, a high-speed CPU. For applications requiring high-speed integer multiplication, the 'ACT8836A can be added. To provide the high precision and large dynamic range of floating point numbers, the 'ACT8847 can be employed. The 'ACT8867 forms the core processor of a graphics subsystem. For multiprocessing applications, the 'ACT8841A can be used to eliminate memory bottlenecks.

To ensure speed and flexibility, each component of the '8800 family has **three data ports**. Each data port accommodates 32 bits of data, plus four parity bits. This architecture eliminates many of the I/O bottlenecks associated with traditional single-I/O microprocessors.

The three-port architecture and functional partitioning of the '8800 chip-set opens the door to a variety of **parallel processing** applications. Placing the math and shifting functions in parallel with the ALU permits concurrent processing of data. Additional processors can be added when performance needs dictate.

The 'ACT8800 building block processors are microprogrammable, so that their instruction sets can be tailored to a specific application. This **high degree of programmability** offers greater speed and flexibility than a typical microprocessor and ensures the most efficient use of hardware.

A **separate control bus** eliminates the need for multiplexing instructions and data, further reducing processing bottlenecks. The microcode bus width is determined by the designer and the application.

Another source of design flexibility is provided by the **pipelined/flowthrough operation option**. Pipelining can dramatically reduce the time required to perform iterative or sequential calculations. On the other hand, random or nonsequential algorithms require low latency operations. The '8800 chipset allows the designer to select the mode (fully pipelined, partially pipelined, or nonpipelined) most suited to each design.

Scientific Accuracy

The '8800 family is designed to support applications that require **double-precision accuracy**. Many scientific applications, such as those in the areas of high-end graphics, digital signal processing, and array processing, require such accuracy to maintain data integrity. In general-purpose computing applications, floating-point processors must often support double-precision data formats to maintain compatibility with existing software.

To ensure data integrity, '8800 devices (excluding the microsequencer) support **parity checking and generation**, as well as **master/slave error detection**. Byte parity checking is performed on the input ports, and a parity generator and a master/slave comparator are provided at the output. **Fault tolerance** is built into the processors, ensuring correct device operation without extra logic or costly software.

The SN74ACT8800 Building Block Processor System

The high-performance '8800 devices are described in the following paragraphs.

SN74ACT8818A 16-Bit Microsequencer

In a high-performance microcoded system, a fast microcode controller is required to control the flow of instructions. The SN74ACT8818A is a high-speed, versatile 16-bit microsequencer capable of addressing 64K words of microcode memory. The 'ACT8818A can address the next instruction fast enough to support a 30-ns system cycle time.

The 'ACT8818A 65-word-deep by 16-bit-wide stack is useful for storing subroutine return addresses, top of loop addresses, and loop counts. Addresses can be sourced from eight different sources: the three I/O ports, the two register counters, the microprogram counter, the stack, and the 16-way branch.

SN74ACT8832A Registered ALU

The SN74ACT8832A is a 32-bit registered ALU that operates at 25 MHz. Because instructions are performed in a single cycle, the 'ACT8832A is capable of executing 25 million microinstructions per second. An on-board 64-word register file is 36-bits-wide (to permit the storage of parity bits). The 3-operand register file increases performance by enabling the creation of an instruction and the storage of the previous result in a single cycle. To facilitate data transfer, operands stored in the register file

can be accessed externally, while the ALU is executing. To support the parallel processing of data, the 'ACT8832A can be configured to operate as four 8-bit ALUs, two 16-bit ALUs, or a single 32-bit ALU. The 'ACT8832A incorporates 32-bit shifters for double-precision shift operations.

SN74ACT8836A 32- × 32-Bit Integer MAC

The SN74ACT8836A is a 32-bit integer multiplier/accumulator (MAC) that accepts two 32-bit inputs and computes a 64-bit product. The device can also operate as a 64-bit by 64-bit multiplier. An onboard adder is provided to add or subtract the product or the complement of the product from the accumulator.

When pipelined internally, the 1- μ m CMOS parallel MAC performs a full 32- × 32-bit multiply/accumulate in a single 30-ns clock cycle. In flowthrough mode (without any pipelining), the 'ACT8836A takes 48 ns to multiply two 32-bit numbers. The 'ACT8836A performs a 64- × 64-bit multiply/accumulate, outputting a 64-bit result, in 194 ns.

The 'ACT8836A can handle a wide variety of data types, including two's complement, signed, and mixed. Division is supported via the Newton-Raphson algorithm.

SN74ACT8841A Digital Crossbar Switch

The SN74ACT8841A is a high-performance digital crossbar switch that cost-effectively eliminates bottlenecks and enables data to speed through complex bus architectures.

The 'ACT8841A is ideal for multiprocessor applications, where memory bottlenecks tend to occur. The device has 64 bidirectional I/O ports that can be configured as 16 4-bit ports, 8 8-bit ports, or 4 16-bit ports. Each bidirectional port can be connected in any conceivable combination. Any single input port can be broadcast to any combination of output ports every 19 ns. The transfer time from data in to data out is 14 ns.

The control sources for ten separate switching configurations are on-chip, including eight banks of programmable control flip-flops and two hard-wired control circuits.

The EPIC™ CMOS SN74ACT8841A and its predecessor, SN74AS8840, are based on the same architecture, differing in power consumption, number of control registers, and pin-out. Microcode written for the 'AS8840 can be run on the 'ACT8841A.

SN74ACT8847 64-Bit Floating-Point and Integer Unit

The SN74ACT8847 is a high-speed 64-bit floating-point and integer unit. The device is fully compatible with IEEE standard 754-1985 for addition, subtraction, multiplication, division, square root, and comparison. Division and square-root operations are implemented via hardwired control.

The 'ACT8847 also performs integer arithmetic, logical operations, and logical shifts. Registers are provided at the inputs, outputs, and inside the ALU and multiplier to support multilevel pipelining. These registers can be bypassed for nonpipelined operations.

When fully pipelined, the 'ACT8847 can perform single-precision floating-point or 32-bit integer operation in 30 ns for 66 MFLOPS (million floating-point operations per second) sustained.

| DEVICE | PACKAGE | CLOCK PERIOD |
|----------------|-------------|--------------|
| SN74ACT8847-30 | 207-pin PGA | 30 ns |
| SN74ACT8847-40 | 207-pin PGA | 40 ns |

SN74ACT8867 32-Bit Vector Processor Unit

The SN74ACT8867 Vector Processor Unit (VPU) is a single-precision (32-bit) floating-point and integer processor optimized for graphics and signal processing. The 32-MFLOPS device is designed to accept either Digital Equipment Corporation's (DEC) floating point 'F' format, 2s complement integers, or 2s complement fractional numbers. The output is either DEC 'F' format, integer format, or a variety of possible fractional numbers with varying radix point placement. All fixed point operands are in 2s complement notation.

The instruction set for the 'ACT8867 is partitioned so that the two processing units, the ALU processor and the multiplier processor, can be operated independently or in parallel.

The 'ACT8867 has two 32-bit input data buses and a 32-bit output data bus. The device supports 40-bit floating point internally and also supports 2s complement integer operations. Forty-six general-purpose registers allow intermediate results to be stored internally. Two other pseudo-register locations, when addressed, serve as a flowthrough function in the register file. In this mode, the register file is bypassed, allowing data to be fed directly from an external source to the processing units.

Single-precision floating point Newton-Raphson division and square root are supported, along with limited support for double-precision arithmetic. Appropriate status is generated for all instructions and data types. The VPU can be operated in flowthrough mode, or pipeline registers are available to allow operation in pipelined mode to increase system speed. All pipeline registers may be placed in a scan mode for device-level and board-level diagnostics.

Bipolar Support Chips

The SN74AS8838 high-speed, 32-bit barrel shifter can shift up to 32 bits in a single instruction cycle of under 25 ns. Five basic shifts can be programmed: circular left, circular right, logical left, logical right, and arithmetic right. The 'AS8838 offloads the responsibility for shifting operations from the ALU, which increases shifter functionality and system throughput.

Glossary

These symbols, terms, and definitions are in accordance with those currently agreed upon by the JEDEC Council of the Electronic Industries Association (EIA) for use in the USA and by the International Electrotechnical Commission (IEC) for international use.

Operating Conditions and Characteristics (in Sequence by Letter Symbols)

f_{max} Maximum clock frequency

The highest rate at which the clock input of a bistable circuit can be driven through its required sequence while maintaining stable transitions of logic level at the output with input conditions established that should cause changes of output logic level in accordance with the specification.

I_{CC} Supply current

The current into[†] the V_{CC} supply terminal of an integrated circuit.

I_{CCQ} Supply current, quiescent

The current into[†] the V_{CC} supply terminal of an integrated circuit when all (or a specified number) of the outputs do not change and the internal logic is in an idle state.

I_{IH} High-level input current

The current into[†] an input when a high-level voltage is applied to that input.

I_{IL} Low-level input current

The current into[†] an input when a low-level voltage is applied to that input.

I_{OH} High-level output current

The current into[†] an output with input conditions applied that, according to the product specification, will establish a high level at the output.

I_{OL} Low-level output current

The current into[†] an output with input conditions applied that, according to the product specification, will establish a low level at the output.

I_{OS} Short-circuit output current

The current into[†] an output when that output is short-circuited to ground (or other specified potential) with input conditions applied to establish the output logic level farthest from ground potential (or other specified potential).

I_{OZ} Off-state (high-impedance-state) output current (of a three-state output)

The current flowing into[†] an output having three-state capability with input conditions established that, according to the production specification, will establish the high-impedance state at the output.

[†]Current out of a terminal is given as a negative value.

t_a Access time

The time interval between the application of a specified input pulse and the availability of valid signals at an output.

t_{dis} Disable time (of a three-state output)

The time interval between the specified reference points on the input and output voltage waveforms, with the three-state output changing from either of the defined active levels (high or low) to a high-impedance (off) state. ($t_{dis} = t_{PHZ}$ or t_{PLZ}).

t_{en} Enable time (of a three-state output)

The time interval between the specified reference points on the input and output voltage waveforms, with the three-state output changing from a high-impedance (off) state to either of the defined active levels (high or low). ($t_{en} = t_{PZH}$ or t_{PZL}).

t_f Fall time

The time interval between two reference points (90% and 10% unless otherwise specified) on a waveform that is changing from the defined high level to the defined low level.

t_h Hold time

The time interval during which a signal is retained at a specified input terminal after an active transition occurs at another specified input terminal.

Notes:

1. The hold time is the actual time interval between two signal events and is determined by the system in which the digital circuit operates. A minimum value is specified that is the shortest interval for which correct operation of the digital circuit is guaranteed.
2. The hold time may have a negative value in which case the minimum limit defines the longest interval (between the release of the signal and the active transition) for which correct operation of the digital circuit is guaranteed.

t_{pd} Propagation delay time

The time between the specified reference points on the input and output voltage waveforms with the output changing from one defined level (high or low) to the other defined level. ($t_{pd} = t_{PHL}$ or t_{PLH}).

t_{PHL} Propagation delay time, high-to-low level output

The time between the specified reference points on the input and output voltage waveforms with the output changing from the defined high level to the defined low level.

t_{PHZ} Disable time (of a three-state output) from high level

The time interval between the specified reference points on the input and the output voltage waveforms with the three-state output changing from the defined high level to a high-impedance (off) state.

- tPLH Propagation delay time, low-to-high-level output**
The time between the specified reference points on the input and output voltage waveforms with the output changing from the defined low level to the defined high level.
- tPLZ Disable time (of a three-state output) from low level**
The time interval between the specified reference points on the input and output voltage waveforms with the three-state output changing from the defined low level to a high-impedance (off) state.
- tpZH Enable time (of a three-state output) to high level**
The time interval between the specified reference points on the input and output voltage waveforms with the three-state output changing from a high-impedance (off) state to the defined high level.
- tpZL Enable time (of a three-state output) to low level**
The time interval between the specified reference points on the input and output voltage waveforms with the three-state output changing from a high-impedance (off) state to the defined low level.
- t_r Rise time**
The time interval between two reference points (10% and 90% unless otherwise specified) on a waveform that is changing from the defined low level to the defined high level.
- t_{su} Setup time**
The time interval between the application of a signal at a specified input terminal and a subsequent active transition at another specified input terminal.
Notes:
1. The setup time is the actual time interval between two signal events and is determined by the system in which the digital circuit operates. A minimum value is specified that is the shortest interval for which correct operation of the digital circuit is guaranteed.
 2. The setup time may have a negative value in which case the minimum limit defines the longest interval (between the active transition and the application of the other signal) for which correct operation of the digital circuit is guaranteed.
- t_t Transition time (general)**
The time interval between two reference points (10% and 90% unless otherwise specified) on a waveform that is changing from the defined low level to the defined high level (rise time) or from the defined high level to the defined low level (fall time).
- t_w Pulse duration (width)**
The time interval between specified reference points on the leading and trailing edges of the pulse waveform.

V_{IH} High-level input voltage

An input voltage within the more positive (less negative) of the two ranges of values used to represent the binary variables.

Note:

A minimum is specified that is the least-positive value of high-level input voltage for which operation of the logic element within specification limits is guaranteed.

V_{IL} Low-level input voltage

An input voltage level within the less positive (more negative) of the two ranges of values used to represent the binary variables.

Note:

A maximum is specified that is the most-positive value of low-level input voltage for which operation of the logic element within specification limits is guaranteed.

V_{OH} High-level output voltage

The voltage at an output terminal with input conditions applied that, according to the product specification, will establish a high level at the output.

V_{OL} Low-level output voltage

The voltage at an output terminal with input conditions applied that, according to the product specification, will establish a low level at the output.

V_{T+} Positive-going threshold level

The voltage level at a transition-operated input that causes operation of the logic element according to specification as the input voltage rises from a level below the negative-going threshold voltage, V_{T-} .

V_{T-} Negative-going threshold level

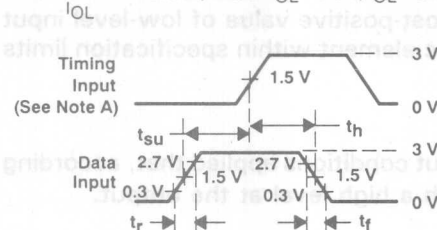
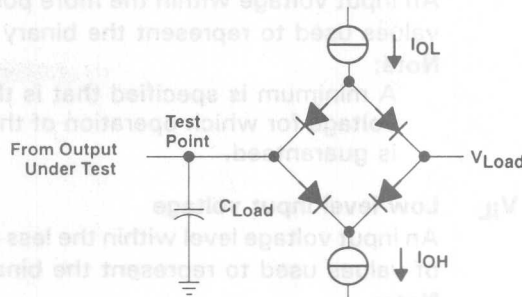
The voltage level at a transition-operated input that causes operation of the logic element according to specification as the input voltage falls from a level above the positive-going threshold voltage, V_{T+} .

PARAMETER MEASUREMENT INFORMATION

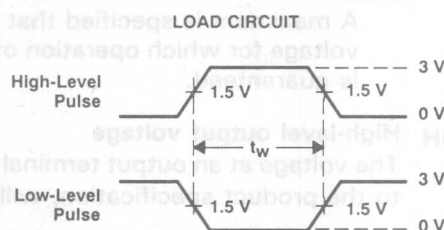
| LOAD CIRCUIT PARAMETERS | | | | | |
|-------------------------|-----------|------------------------------|------------------|------------------|-------------------|
| TIMING PARAMETERS | | C_{LOAD}^{\dagger} (pF) | I_{OL} (mA) | I_{OH} (mA) | V_{LOAD} (V) |
| t_{en} | t_{PZH} | 50 | 8 | -8 | 0 |
| | t_{PZL} | | | | 3 |
| t_{dis} | t_{PHZ} | 50 | 8 | -8 | 1.5 |
| | t_{PLZ} | | | | 1.5 |
| t_{pd} | | 50 | 8 | -8 | ‡ |

$^{\dagger} C_{LOAD}$ includes probes and test fixture capacitance.

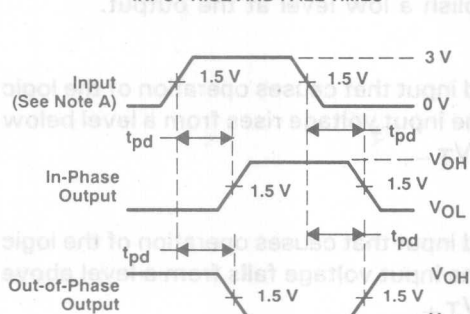
$^{\ddagger} V_{LOAD} - V_{OL} = 50 \Omega$, where $V_{OL} = 0.6 V$, $I_{OL} = 8 mA$.



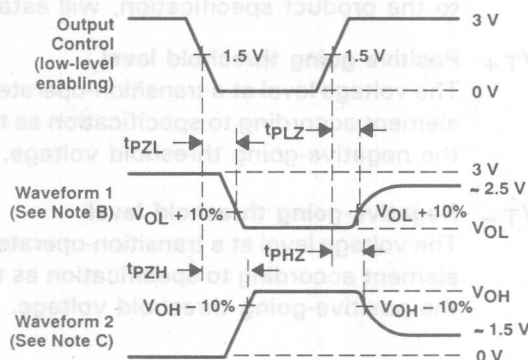
VOLTAGE WAVEFORMS
SETUP AND HOLD TIMES
INPUT RISE AND FALL TIMES



VOLTAGE WAVEFORMS
PULSE DURATION



VOLTAGE WAVEFORMS
PROPAGATION DELAY TIMES



VOLTAGE WAVEFORMS
ENABLE AND DISABLE TIMES, 3-STATE OUTPUTS

- NOTES: A. Phase relationships between waveforms were chosen arbitrarily. All input pulses are supplied by pulse generators having the following characteristics: $PRR = 1 MHz$, $Z_O = 50 \Omega$, $t_r \leq 6 ns$, $t_f \leq 6 ns$.
- B. Waveform 1 is for an output with internal conditions such that the output is low except when disabled by the output control.
- C. Waveform 2 is for an output with internal conditions such that the output is high except when disabled by the output control. For t_{PLZ} and t_{PHZ} , V_{OL} and V_{OH} are specified values.

Figure 1

| | |
|---|-----------|
| General Information | 1 |
| SN74ACT8818A 16-Bit Microsequencer | 2 |
| SN74ACT8832A 32-Bit Registered ALU | 3 |
| SN74ACT8836A 32- × 32-Bit Parallel Multiplier | 4 |
| SN74ACT8841A Digital Crossbar Switch | 5 |
| SN74ACT8847 64-Bit Floating-Point/Integer Unit | 6 |
| SN74ACT8847 Application Information | 7 |
| SN74ACT8867 32-Bit Vector Processor Unit | 8 |
| Design Support | 9 |
| Mechanical Data | 10 |

1 **16-Bit Microsequencer**

2 **16-Bit Microsequencer**

3 **16-Bit Microsequencer**

4 **16-Bit Microsequencer**

5 **16-Bit Microsequencer**

6 **16-Bit Microsequencer**

7 **16-Bit Microsequencer**

8 **16-Bit Microsequencer**

9 **16-Bit Microsequencer**

10 **16-Bit Microsequencer**

SN74ACT8818A 16-BIT MICROSEQUENCER

D3471, JANUARY 1990

- Addresses Up to 64K Locations of Microprogram Memory
- CLK-to-Y = 24 ns Max (t_{pd})
- Low-Power 1- μ m EPIC™ CMOS
- Addresses Selected from Eight Different Sources
- Performs Multiway Branching, Conditional Subroutine Calls, and Nested Loops
- Large 64-Word by 16-bit Stack
- Cascadable

description

The SN74ACT8818A microsequencer is a low-power, high-performance microsequencer implemented in TI's EPIC™ Advanced CMOS technology. The 16-bit device addresses up to 64K locations of microprogram memory and is compatible with the SN74AS890 microsequencer.

The 'ACT8818A performs a range of sequencing operations in support of TI's family of building block devices and special-purpose processors such as the SN74ACT8847 Floating Point Unit (FPU).

The 'ACT8818A microsequencer is designed to control execution of microcode in a microprogrammed system. Basic architecture of such a system usually incorporates at least the microsequencer, one or more processing elements such as the 'ACT8847 FPU or the SN74ACT8832 registered ALU, microprogram memory, microinstruction register, and status logic to monitor system states and provide status inputs to the microsequencer.

The 'ACT8818A combines flexibility and high speed in a microsequencer that performs multiway branching, conditional subroutine calls, nested loops, and a variety of other microprogrammable operations. The 'ACT8818A can also be cascaded for providing additional register/counters or addressing capability for more complex microcoded control functions.

In this microsequencer, several sources are available for microprogram address selection. The primary source is the 16-bit microprogram counter (MPC), although branch addresses may be input on the two 16-bit address buses, DRA and DRB. An address input on the DRA bus can be pushed on the stack for later selection. Register/counters RCA and RCB can store either branch addresses or loop counts as needed, either for branch operations or for looping on the stack.

The selection of address source can be based on external status from the device being controlled, so that three-way or multiway branching is supported. Once selected, the address which is output on the Y bus passes to the microprogram memory, and the microinstruction from the selected location is clocked into the pipeline register at the beginning of the next cycle.

It is also possible to interrupt the 'ACT8818A by placing the Y output bus in a high-impedance state and forcing an interrupt vector on the Y bus. External logic is required to place the bus in high impedance and load the interrupt vector. The first microinstruction of the interrupt handler subroutine can push the address from the Interrupt Return register on the stack so that proper linkage is preserved for the return from subroutine.

Microinstructions for the 'ACT8818A select the specific operations performed by the Y output multiplexer, the register/counters RCA and RCB, the stack, and the bidirectional DRA and DRB buses. Each set of inputs is represented as a separate field in the microinstructions, which control not only the microsequencer but also the ALU or other devices in the system.

The 3-port architecture of the 'ACT8818A facilitates both branch addressing and register/counter operations. Both register/counters can be used to hold either loop counts or branch addresses loaded from the DRA and DRB buses. Register/counter operations are selected by control inputs RC2-RC0.

EPIC is a trademark of Texas Instruments Incorporated.

PRODUCTION DATA documents contain information current as of publication date. Products conform to specifications per the terms of Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.

TEXAS
INSTRUMENTS

POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

SN74ACT8818A 16-BIT MICROSEQUENCER

description (continued)

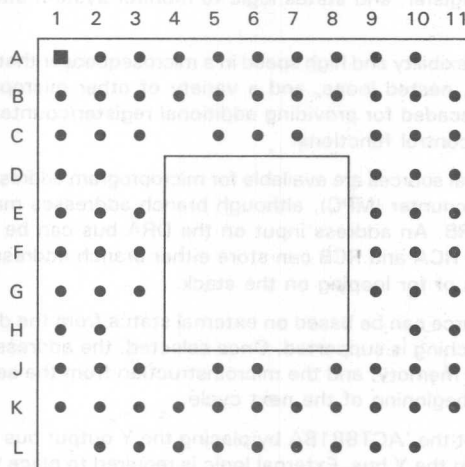
Similarly, the 65-word by 16-bit stack can save addresses from the DRA bus, the microprogram counter (MPC), or the Interrupt Return register, depending on the settings of stack controls S2-S0 and related control inputs. Flexible instructions such as Branch DRA else Branch to Stack else Continue can be coded to take advantage of the conditional branching capability of the 'ACT8818A.

Multiway branching (16- or 32-way) uses the B3-B0 inputs to set up a 16-way branch address on DRA or DRB by concatenating B3-B0 with the upper 12 bits of the DRA or DRB bus. The resulting branch addresses DRA' (DRA15-DRA4::B3-B0) and DRB' (DRB15-DRB4::B3-B0) are selected by the Y output multiplexer controls MUX2-MUX0. A Branch DRB' else Branch DRA' instruction can select up to 32 branch addresses, as determined by the settings of B3-B0.

pin description

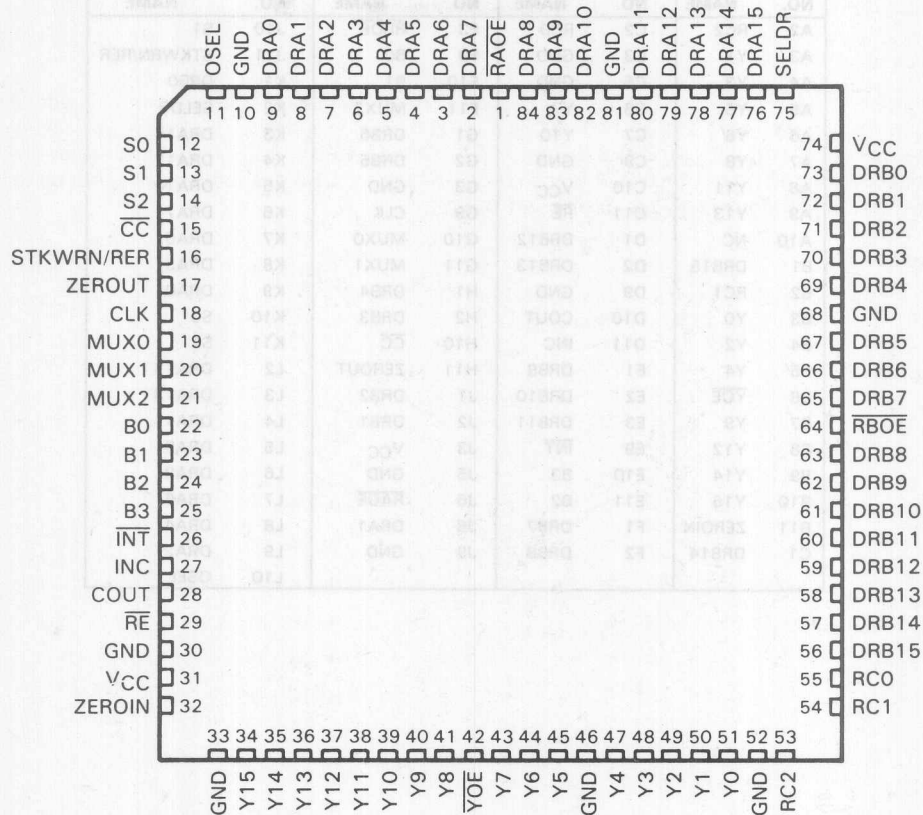
The pin descriptions and grid assignments for the 'ACT8818A are given on the following pages. The grid location A1 is marked for indexing purposes.

GC PACKAGE
(TOP VIEW)



SN74ACT8818A
16-BIT MICROSEQUENCER

FN PACKAGE
(TOP VIEW)



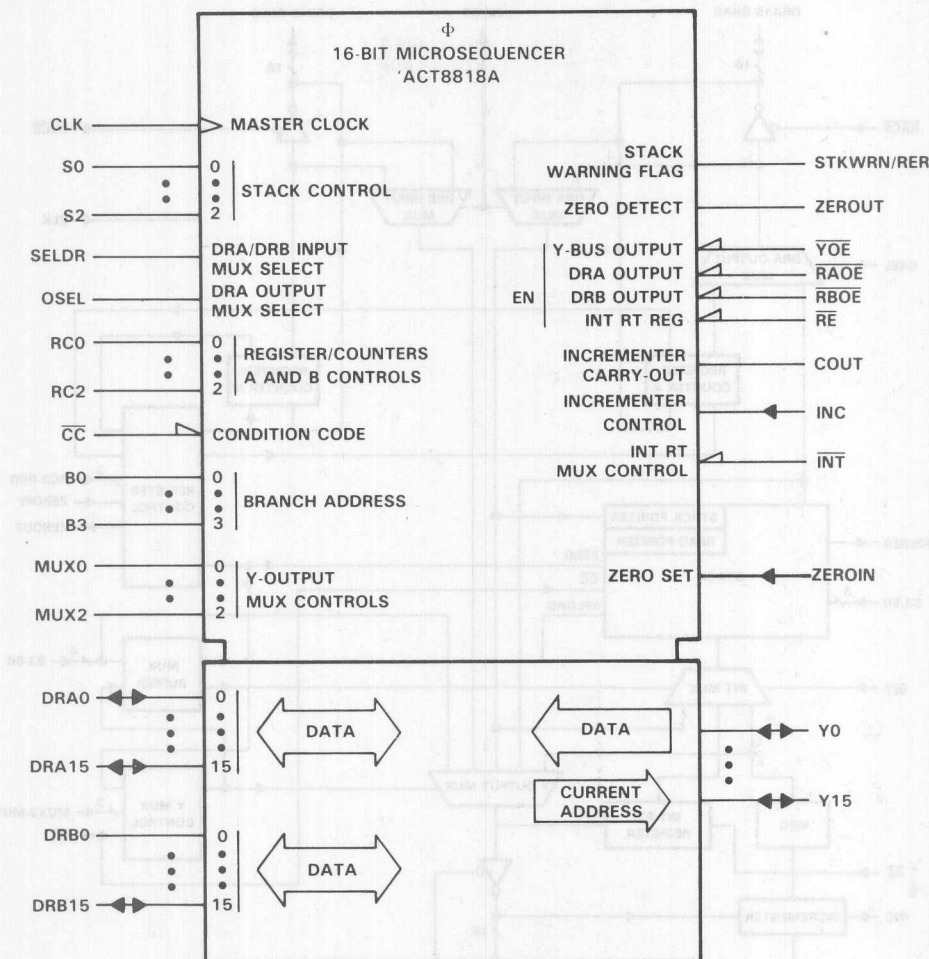
SN74ACT8818A
16-BIT MICROSEQUENCER

Pin Assignments, GC Package

| PIN | | PIN | | PIN | | PIN | |
|-----|--------|-----|-----------------|-----|-----------------|-----|------------|
| NO. | NAME | NO. | NAME | NO. | NAME | NO. | NAME |
| A2 | RC2 | C2 | RC0 | F3 | RBOE | J10 | S1 |
| A3 | Y1 | C3 | GND | F9 | B0 | J11 | STKWRN/RER |
| A4 | Y3 | C5 | GND | F10 | B1 | K1 | DRB0 |
| A5 | Y5 | C6 | Y7 | F11 | MUX2 | K2 | SELDL |
| A6 | Y6 | C7 | Y10 | G1 | DRB6 | K3 | DRA14 |
| A7 | Y8 | C9 | GND | G2 | DRB5 | K4 | DRA12 |
| A8 | Y11 | C10 | V _{CC} | G3 | GND | K5 | DRA10 |
| A9 | Y13 | C11 | RE | G9 | CLK | K6 | DRA7 |
| A10 | NC | D1 | DRB12 | G10 | MUX0 | K7 | DRA5 |
| B1 | DRB15 | D2 | DRB13 | G11 | MUX1 | K8 | DRA3 |
| B2 | RC1 | D9 | GND | H1 | DRB4 | K9 | DRA0 |
| B3 | Y0 | D10 | COUT | H2 | DRB3 | K10 | S0 |
| B4 | Y2 | D11 | INC | H10 | CC | K11 | S2 |
| B5 | Y4 | E1 | DRB9 | H11 | ZEROUT | L2 | DRA15 |
| B6 | YOE | E2 | DRB10 | J1 | DRB2 | L3 | DRA13 |
| B7 | Y9 | E3 | DRB11 | J2 | DRB1 | L4 | DRA11 |
| B8 | Y12 | E9 | INT | J3 | V _{CC} | L5 | DRA9 |
| B9 | Y14 | E10 | B3 | J5 | GND | L6 | DRA8 |
| B10 | Y15 | E11 | B2 | J6 | RAOE | L7 | DRA6 |
| B11 | ZEROIN | F1 | DRB7 | J8 | DRA1 | L8 | DRA4 |
| C1 | DRB14 | F2 | DRB8 | J9 | GND | L9 | DRA2 |
| | | | | | | L10 | OSEL |

SN74ACT8818A 16-BIT MICROSEQUENCER

logic symbol[†]



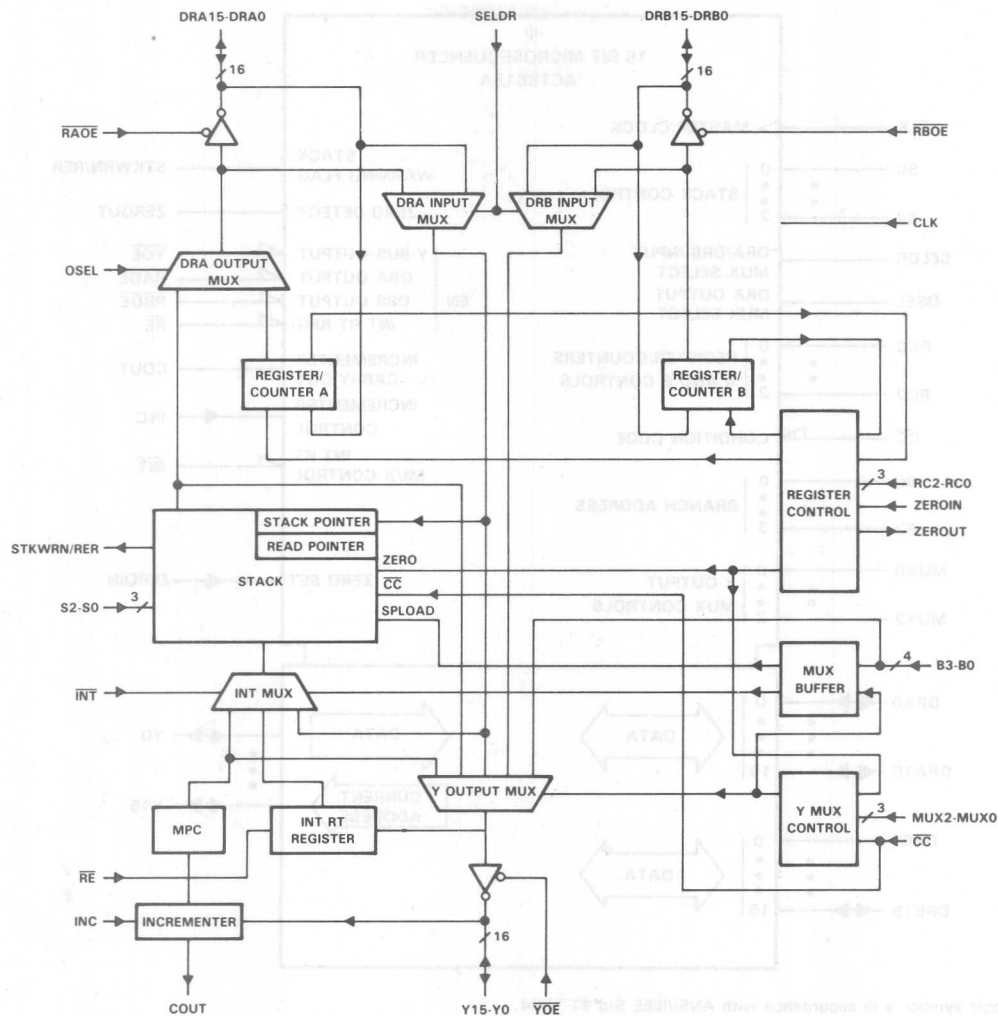
[†]This logic symbol is in accordance with ANSI/IEEE Std 91-1984.

TEXAS
INSTRUMENTS

POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

SN74ACT8818A **16-BIT MICROSEQUENCER**

functional block diagram



SN74ACT8818A 16-BIT MICROSEQUENCER

TERMINAL FUNCTIONS

| NAME | PIN FN NO. | GC NO. | I/O | DESCRIPTION |
|---------------|------------------|-----------|-----|---|
| B0 | 22 | F9 | I | Input bits for branch addressing (see Table 3) |
| B1 | 23 | F10 | | |
| B2 | 24 | E11 | | |
| B3 | 25 | E10 | | |
| CLK | 18 | G9 | | System clock |
| COU \bar{T} | 28 | D10 | O | Incrementer carry-out. Goes high when an attempt is made to increment microprogram counter beyond addressable micromemory. |
| $\bar{C}C$ | 15 | H10 | I | Condition code |
| DRA0 | 9 | K9 | I/O | Bidirectional DRA data port. Outputs data from stack or register/counter A ($\bar{R}AOE = 0$) or inputs external data ($\bar{R}AOE = 1$). |
| DRA1 | 8 | J8 | | |
| DRA2 | 7 | L9 | | |
| DRA3 | 6 | K8 | | |
| DRA4 | 5 | L8 | | |
| DRA5 | 4 | K7 | | |
| DRA6 | 3 | L7 | | |
| DRA7 | 2 | K6 | | |
| DRA2 | 7 | L9 | | |
| DRA9 | 83 | L5 | | |
| DRA10 | 82 | K5 | | |
| DRA11 | 80 | L4 | | |
| DRA12 | 79 | K4 | | |
| DRA13 | 78 | L3 | | |
| DRA14 | 77 | K3 | | |
| DRA15 | 76 | L2 | | |
| DRB0 | 73 | K1 | I/O | Bidirectional DRB data port. Outputs data from register/counter B ($\bar{R}BOE = 0$) or inputs external data ($\bar{R}BOE = 1$). |
| DRB1 | 72 | J2 | | |
| DRB2 | 71 | J1 | | |
| DRB3 | 70 | H2 | | |
| DRB4 | 69 | H1 | | |
| DRB5 | 67 | G2 | | |
| DRB6 | 66 | G1 | | |
| DRB7 | 65 | F1 | | |
| DRB8 | 63 | F2 | I/O | Bidirectional DRB data port. Outputs data from register/counter B ($\bar{R}BOE = 0$) or inputs external data ($\bar{R}BOE = 1$). |
| DRB10 | 61 | E2 | | |
| DRB11 | 60 | E3 | | |
| DRB12 | 59 | D1 | | |
| DRB13 | 58 | D2 | I/O | Bidirectional DRB data port. Outputs data from register/counter B ($\bar{R}BOE = 0$) or inputs external data ($\bar{R}BOE = 1$). |
| DRB14 | 57 | C1 | | |
| DRB15 | 56 | B1 | | |
| GND | 10 | C3 | | Ground pins. All pins must be used. |
| GND | 30 | C5 | | |
| GND | 33 | C9 | | |
| GND | 46 | D9 | | |
| GND | 52 | G3 | | |
| GND | 68 | J5 | | |
| GND | 81 | J9 | | |
| INC | 27 | D11 | I | Incrementer control pin |

SN74ACT8818A 16-BIT MICROSEQUENCER

TERMINAL FUNCTIONS (continued)

| NAME | PIN FN NO. | GC NO. | I/O | DESCRIPTION |
|----------------|------------------|-----------|-----|---|
| INT | 26 | E9 | I | Selects INT RT register to stack, active low (see Table 3) |
| MUX0 | 9 | G10 | | |
| MUX1 | 20 | G11 | I | MUX control for Y output bus (see Table 4) |
| MUX2 | 21 | F11 | | |
| OSEL | 11 | L10 | I | DRA output MUX select. Low selects RCA, high selects stack. |
| RAOE | 1 | J6 | I | DRA output enable, active low |
| RBOE | 64 | F3 | I | DRB output enable, active low |
| RC0 | 55 | C2 | | |
| RC1 | 54 | B2 | I | Controls for register/counters A and B |
| RC2 | 53 | A2 | | |
| RE | 29 | C11 | | INT RT register enable, active low. A high input holds INT RT register while a low input passes Y to INRT register (see Table 3). |
| S0 | 12 | K10 | | |
| S1 | 13 | J10 | I | Stack controls |
| S2 | 14 | K11 | | |
| SELDRA | 75 | K2 | I | Selects data source to DRA bus and DRB bus (See Table 3) |
| STKWRN/ RER | 16 | J11 | O | Stack warning signal flag |
| VCC | 31 | C10 | | |
| VCC | 74 | J3 | | Supply voltage (5 V) |
| Y0 | 51 | B3 | | |
| Y1 | 50 | A3 | | |
| Y2 | 49 | B4 | | |
| Y3 | 48 | A4 | | |
| Y4 | 47 | B5 | | |
| Y5 | 45 | A5 | | |
| Y6 | 44 | A6 | | |
| Y7 | 43 | C6 | I/O | Bidirectional Y data port |
| Y8 | 41 | A7 | | |
| Y9 | 40 | B7 | | |
| Y10 | 39 | C7 | | |
| Y11 | 38 | A8 | | |
| Y12 | 37 | B8 | | |
| Y13 | 36 | A9 | | |
| Y14 | 35 | B9 | | |
| Y15 | 34 | B10 | | |
| YOE | 42 | B6 | I | Y output enable, active low |
| ZEROIN | 32 | B11 | I | Forces internal zero detect high |
| ZEROUT | 17 | H11 | O | Outputs register/counter zero detect signal |

design support

TI's 'ACT8818A 16-bit microsequencer is supported by a variety of tools developed to aid in design evaluation and verification. These tools will streamline all stages of the design process, from assessing the operation and performance of the 'ACT8818A to evaluating a total system application. The tools include a functional model, behavioral model, and microcode development software and hardware. Section 9 of this manual provides specific information on the design tools supporting TI's '8800 Family.

systems expertise

Texas Instruments Datapath VLSI Products Systems Engineering group is available to help designers analyze TI's high-performance VLSI products, such as the 'ACT8818A 16-bit microsequencer. The group works directly with designers to provide ready answers to device-related questions and also prepares a variety of applications documentation.

The group may be reached in Dallas, at (214) 997-3970.

architecture

The 'ACT8818A microsequencer is designed with a 3-port architecture similar to the bipolar SN74AS890 microsequencer. The functional block diagram shows the architecture of the 'ACT8818A. The device consists of the following principal functional groups:

1. A 16-bit microprogram counter (MPC) consisting of a register and incrementer which generates the next sequential microprogram address
2. Two register/counters (RCA and RCB) for counting loops and iterations, storing branch addresses, or driving external devices
3. A 65-word by 16-bit LIFO stack that allows subroutine calls and interrupts at the microprogram level and is expandable and readable by external hardware
4. An interrupt return register and Y output enable for interrupt processing at the microinstruction level
5. A Y output multiplexer by which the next address can be selected from MPC, RCA, RCB, external buses DRA and DRB, or the stack.

'ACT8818A control signals are summarized in Table 1. Those signals, which typically originate from the instruction register, are Y output multiplexer controls, MUX2-MUX0. These select the source of the next address; stack operation controls, S2-S0; register/counter operation controls, RC2-RC0; OSEL, which allows the stack to be read for diagnostics; input MUX select, SELDR; DRA and DRB output enables, \overline{RAOE} and \overline{RBOE} ; and \overline{INT} , used during the first cycle of interrupt service routines to push the address in the interrupt return register address onto the stack.

Control and data signals that commonly originate from the microinstruction and from other hardware sources include INC, which determines whether to increment the MPC; DRA and DRB, used to load or read loop counters and/or next addresses; and \overline{CC} , the condition code input. The address being loaded into the MPC is not incremented if INC is low, allowing wait states and repeat until flag instructions to be implemented. If INC originates from status, repeat until flag instructions are possible.

The condition code input \overline{CC} typically originates from ALU status to permit test and branch instructions. However, it must also be asserted under microprogram control to implement other instructions such as continue or loop. Therefore, \overline{CC} will generally be controlled by the output of a status multiplexer. In this case, whether \overline{CC} is to be forced high, forced low, or taken from ALU status will be determined by a status MUX select field in the microinstruction.

Control signals that may also originate from hardware are B3-B0, which can be used as a 4-bit status input to support 16- and 32-way branches, and YOE, which allows interrupt hardware to force an interrupt vector on the microaddress bus.

Status from the 'ACT8818A is provided by ZEROOUT, which is set at the beginning of a cycle in which either of the register/counters will decrement to zero, and STKWRN/RER, set at the beginning of the cycle in which the bottom of stack is read or in which the next to last location is written. In the latter case, STKWRN/RER remains high until the stack pointer is decremented from 64 to 63.

SN74ACT8818A

16-BIT MICROSEQUENCER

Table 1. Response to Control Inputs

| SIGNAL NAME | LOGIC LEVEL | |
|--------------------------------------|--|--|
| | HIGH | LOW |
| B0 [†] | Load stack pointer from 7 least significant bits of DRA | No effect |
| B1 [†] | Selects DRA contents as stack input (takes priority over $\overline{\text{INT}}$) | No effect |
| $\overline{\text{CC}}$ | Condition code input. May be microcoded or selected from external status results. | Condition code input. For branch operations, low active. |
| INC | Increment address from Y bus and load into MPC | Pass address from Y bus to MPC unincremented. |
| $\overline{\text{INT}}$ [‡] | Selects MPC as input to stack | Selects interrupt return register as input to stack |
| OSEL | Selects stack as output from DRA output MUX | Selects RCA as output from DRA output MUX |
| MUX2-MUX0 | See Table 4 | See Table 4 |
| $\overline{\text{RAOE}}$ | DRA output disabled (high-Z) | DRA output enabled |
| $\overline{\text{RBOE}}$ | DRB output disabled (high-Z) | DRB output enabled |
| RC2-RC0 | See Table 6 | See Table 6 |
| $\overline{\text{RE}}$ | Hold interrupt return register contents | Load address on Y bus to interrupt return register |
| S2-S0 | See Table 5 | See Table 5 |
| SELDR | Selects DRA/DRB external data as inputs to DRA/DRB buses | Selects RCA (OSEL low) or stack (OSEL high) to DRA bus, RCB to DRB bus |
| $\overline{\text{YOE}}$ | Y output disabled (high-Z) | Y output enabled |
| ZEROIN | Sets ZEROOUT to a high externally to set up conditional branch | No effect |

[†]No control effect when DRA' or DRB' selected (MUX2-MUX0) = HLH) because B3-B0 are address inputs.

[‡]When B1 is low or B1 is not in control mode.

Y output multiplexer

Address selection is controlled by the Y output multiplexer and the $\overline{\text{RAOE}}$ and $\overline{\text{RBOE}}$ enables. Addresses can be selected from eight sources:

1. The microprogram counter register, used for repeat (INC off) and continue (INC on) instructions
2. The stack, which supports subroutine calls and returns as well as iterative loops and returns from interrupts
3. The DRA and DRB ports, which provide two additional paths from external hardware by which microprogram addresses can be generated
4. Register counters RCA and RCB, which can be used for additional address storage
5. B3-B0, whose contents can replace the four least significant bits of the DRA and DRB buses to support 16-way and 32-way branches
6. An external input onto the bidirectional Y port to support external interrupts.

Use of controls MUX2-MUX0 is explained further in the later section on microprogramming the 'ACT8818A.

microprogram counter

Based on system status and the current instruction, the microsequencer outputs the next execution address in the microprogram. Usually the incrementer adds one to the address on the Y bus to compute next address plus one. Next address plus one is stored in the microprogram register at the beginning of the subsequent instruction cycle. During the next instruction, this 'continue' address will be ready at the Y output MUX for possible selection as the source of the subsequent instruction. The incrementer thus looks two addresses ahead of the address in the instruction register to set up a continue (increment by one) or repeat (no increment) address.

Selecting INC from status is a convenient means of implementing instructions that must repeat until some condition is satisfied; for example, Shift ALU Until MSB = 1, or Decrement ALU Until Zero. The MPC is also the standard path to the stack. The next address is pushed onto the stack during a subroutine call, so that the subroutine will return to the instruction following that from which it was called.

register/counters

Addresses or loop counts may be loaded directly into register/counters RCA and RCB through the direct data ports DRA15-DRA0 and DRB15-DRB0. The values stored in these registers may either be held, decremented, or read. Independent control of both the registers during a single cycle is supported with the exception of a simultaneous decrement of both registers.

stack

The positive-edge-clocked 16-bit address stack allows multiple levels of nested calls or interrupts and can be used to support branching and looping. Seven stack operations are possible:

1. Reset, which pulls all Y outputs low and clears the stack pointer and read pointer
2. Clear, which sets the stack pointer and read pointer to zero
3. Pop, which causes the stack pointer to be decremented
4. Push, which puts the contents of the MPC, interrupt return register, or DRA bus onto the stack and increments the stack pointer
5. Read, which makes the address indicated by the read pointer available at the DRA port
6. Hold, which causes the address of the stack and read pointers to remain unchanged
7. Load stack pointer, which inputs the seven least significant bits of DRA to the stack pointer.

stack pointer

The stack pointer (SP) operates as an up/down counter; it increments whenever a push occurs and decrements whenever a pop occurs. Although push and pop are two event operations (store then increment SP, or decrement SP then read), the 'ACT8818A performs both events within a single cycle.

read pointer

The read pointer (RP) is provided as a tool for debugging microcoded systems. It permits a nondestructive, sequential read of the stack contents from the DRA port. This capability provides the user with a method of backtracking through the address sequence to determine the cause of overflow without affecting program flow, the status of the stack pointer, or the internal data of the stack.

SN74ACT8818A

16-BIT MICROSEQUENCER

stack warning/read error pin

A high signal on the STKWRN/RER pin indicates a potential stack overflow or underflow condition. STKWRN/RER becomes active under two conditions. If 62 of the 65 stack locations (0-64) are full (the stack pointer is at 62) and a push occurs, the STKWRN/RER pin outputs a high signal to warn that the stack is approaching its capacity and will be full after two more pushes.

The STKWRN/RER signal will remain high if hold, push, or pop instructions occur, until the stack pointer is decremented to 62. If a push instruction is attempted when the stack is full, the new address will be ignored and the old address in stack location 64 will be retained.

The STKWRN/RER pin will go high when the stack pointer is less than or equal to one and a pop or read from stack is coded on the S2-S0 pins. The pin will go high after reading the next to the bottom stack address (1). When the S2-S0 pins are set to pop or read the last address (0) or to pop or read an empty stack, the STKWRN/RER pin will go high. The pin depends only on the setting of the S2-S0 pins and the stack pointer, not on the clock.

interrupt return register

Unlike the MPC register, which normally gets next address plus one, the interrupt return register simply gets next address. This permits interrupts to be serviced with zero latency, since the interrupt vector replaces the pending address.

The interrupting hardware disables the Y output and forces the vector onto the microaddress bus. This event must be synchronized with the system clock. The first address of the service routine must program INT low and perform a push to put the contents of the interrupt return register on the stack.

absolute maximum ratings over operating free air temperature range (unless otherwise noted)†

| | |
|--|---------------|
| Supply voltage, V_{CC} (see Note 1) | -0.5 V to 6 V |
| Input clamp current, I_{IK} ($V_I < 0$ or $V_I > V_{CC}$) | ± 20 mA |
| Output clamp current, I_{OK} ($V_O < 0$ or $V_O > V_{CC}$) | ± 50 mA |
| Continuous output current, I_O ($V_O = 0$ to V_{CC}) | ± 50 mA |
| Continuous current through V_{CC} or GND pins | ± 100 mA |
| Operating free-air temperature range | 0°C to 70°C |
| Storage temperature range | 65°C to 150°C |

†Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute maximum rated conditions for extended periods may affect device reliability.

NOTE 1: All voltage levels are with respect to GND.

recommended operating conditions

| | MIN | NOM | MAX | UNIT |
|--|------|-----|----------|------|
| V_{CC} Supply voltage | 4.75 | 5 | 5.25 | V |
| V_{IH} High-level input voltage | 2 | | V_{CC} | V |
| V_{IL} Low-level input voltage | 0 | | 0.8 | V |
| I_{OH} High-level output current | | | -8 | mA |
| I_{OL} Low-level output current | | | 8 | mA |
| V_I Input voltage | 0 | | V_{CC} | V |
| V_O Output voltage | 0 | | V_{CC} | V |
| dt/dv Input transition rise or fall rate | 0 | | 15 | ns/V |
| TA Operating free-air temperature | 0 | | 70 | °C |

electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

| PARAMETER | TEST CONDITIONS | V _{CC} | MIN | TYP [†] | MAX | UNIT |
|-------------------------------|--|-----------------|------|------------------|------|------|
| V _{OH} | I _{OH} = -20 µA | 4.75 V | 4.6 | | | V |
| | | 5.25 V | 5.1 | | | |
| | I _{OH} = -8 mA | 4.75 V | 3.85 | 3.95 | | |
| | | 5.25 V | 4.6 | 4.7 | | |
| V _{OL} | I _{OL} = 20 µA | 4.75 V | | | 0.1 | V |
| | | 5.25 V | | | 0.1 | |
| | I _{OL} = 8 mA | 4.75 V | | 0.32 | 0.45 | |
| | | 5.25 V | | 0.32 | 0.45 | |
| I _I [‡] | V _I = V _{CC} or 0 | 5.25 V | ±0.1 | | ±5 | µA |
| I _{CCQ} | V _I = V _{CC} or 0 | 5.25 V | | 100 | 200 | µA |
| C _I | V _I = V _{CC} or 0 | 5 V | | 10 | | pF |
| ΔI _{CC} [§] | One input at 3.4 V, other inputs at 0 or V _{CC} | 5.25 V | | | 1 | mA |

[†]All typical values are at V_{CC} = 5 V, T_A = 25°C.

[‡]For I/O ports, the parameter I_{OZH} and I_{OZL} include the offstate output current.

[§]This is the increase in supply current for each input that is at one of the specified TTL voltage levels rather than 0 V or V_{CC}.

maximum switching characteristics over recommended operating supply voltage and free-air temperature range (unless otherwise noted)[†]

| PARAMETER | FROM (INPUT) | TO (OUTPUT) | | | | | | UNIT |
|------------------|-----------------|----------------|--------|-----|-----|--------|------|------|
| | | Y | ZEROUT | DRA | DRB | STKWRN | COUT | |
| | CC | 17 | | | | | | ns |
| | CLK | 22 24# | 17# | 18 | 12 | 19 | | ns |
| t _{pd} | DRA15-DRA0 | 17 | | | | | | ns |
| | DRB15-DRB0 | 17 | | | | | | |
| | MUX2-MUX0 | 17 | | | | | | |
| | RC2-RC0 | 19 | 14 | | | | | |
| | S2-S0 | 18 | | 14 | | | | |
| | B3-B0 | 14 | | | | | | |
| | OSEL | 18 | | 15 | | | | |
| | ZEROIN | 18 | | | | | | |
| | SELDR | 16 | | | | | | |
| | INC | | | | | | 15 | |
| t _{en} | Y | | | | | | 13 | ns |
| | YOE | 16 | | | | | | |
| | RAOE | | | 18 | | | | |
| t _{dis} | RBOE | | | | 17 | | | ns |
| | YOE | 12 | | | | | | |
| | RAOE | | | 12 | | | | |
| | RBOE | | | | 12 | | | |

[†]See Parameter Measurement Information for load circuit and voltage waveforms.

#Decrementing register/counter A or B and sensing a zero.



POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

SN74ACT8818A 16-BIT MICROSEQUENCER

timing requirements over recommended ranges of supply voltage and operating free-air temperature
(unless otherwise noted)

setup and hold times

| | PARAMETER | MIN | MAX | UNIT |
|-----------------|---|-----|-----|------|
| V | Setup time, from \overline{CC} to CLK, destination is stack | 11 | | ns |
| | Setup time, from DRA15-DRA0 to CLK1, destination is stack | 9 | | ns |
| | Setup time, from DRA15-DRA0 to CLK1, destination is RCA | 6 | | ns |
| V | Setup time, from DRA15-DRA0 to CLK1, destination is INT RT | 11 | | ns |
| | Setup time, from DRB15-DRB0 to CLK1, destination is RCB | 7 | | ns |
| | Setup time, from DRB15-DRB0 to CLK1, destination is INT RT | 11 | | ns |
| V | Setup time, from INC to CLK1, destination is MPC | 11 | | ns |
| | Setup time, from \overline{INT} to CLK1, destination is stack | 7 | | ns |
| | Setup time, from RC2-RC0 to CLK1, destination is stack | 7 | | ns |
| V | Setup time, from RC2-RC0 to CLK1, destination is RCA or RCB | 14 | | ns |
| | Setup time, from RC2-RC0 to CLK1, destination is INT RT | 6 | | ns |
| t _{su} | Setup time, from S2-S0 to CLK1, destination is stack | 11 | | ns |
| | Setup time, from S2-S0 to CLK1, destination is INT RT | 11 | | ns |
| | Setup time, from OSEL to CLK1, destination is stack | 10 | | ns |
| t _{su} | Setup time, from OSEL to CLK1, destination is INT RT | 10 | | ns |
| | Setup time, from B3-B0 to CLK1, destination is stack | 8 | | ns |
| | Setup time, from B3-B0 to CLK1, destination is INT RT | 11 | | ns |
| t _{su} | Setup time, from SELDR to CLK1, destination is stack | 8 | | ns |
| | Setup time, from SELDR to CLK1, destination is INT RT | 8 | | ns |
| t _{su} | Setup time, from ZEROIN to CLK1, destination is stack | 12 | | ns |
| | Setup time, from ZEROIN to CLK1, destination is INT RT | 10 | | ns |
| t _{su} | Setup time, from Y to CLK1, destination is MPC | 6 | | ns |
| | Setup time, from \overline{RE} to CLK1, destination is INT RT | 6 | | ns |
| | Setup time, from MUX2-MUX0 to CLK1, destination is INT RT | 9 | | ns |
| t _h | Hold time, any input after CLK1 | 0 | | ns |

clock requirements

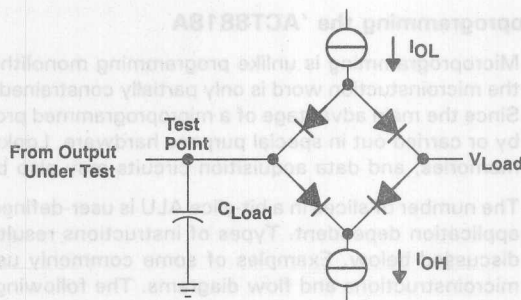
| | PARAMETER | MIN | MAX | UNIT |
|-----------------|----------------------------|-----|-----|------|
| t _{w1} | Pulse duration, clock low | 8 | | ns |
| t _{w2} | Pulse duration, clock high | 8 | | ns |
| t _c | Clock cycle time | 25 | | ns |

PARAMETER MEASUREMENT INFORMATION

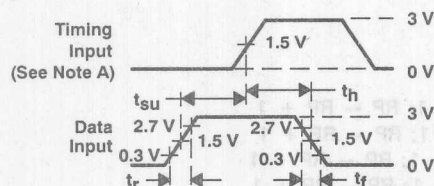
| LOAD CIRCUIT PARAMETERS | | | | | |
|-------------------------|------------------|-------------------------------------|----------------------|----------------------|-----------------------|
| TIMING PARAMETERS | | C _{LOAD} [†] (pF) | I _{OL} (mA) | I _{OH} (mA) | V _{LOAD} (V) |
| t _{en} | t _{PZH} | 50 | 8 | -8 | 0 |
| | t _{PZL} | | | | 3 |
| t _{dis} | t _{PHZ} | 50 | 8 | -8 | 1.5 |
| | t _{PLZ} | | | | 1.5 |
| t _{pd} | | 50 | 8 | -8 | ‡ |

[†] C_{LOAD} includes probes and test fixture capacitance.

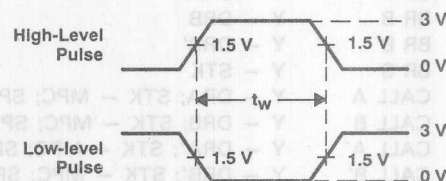
[‡] $\frac{V_{LOAD} - V_{OL}}{I_{OL}} = 50 \Omega$, where V_{OL} = 0.6 V, I_{OL} = 8 mA.



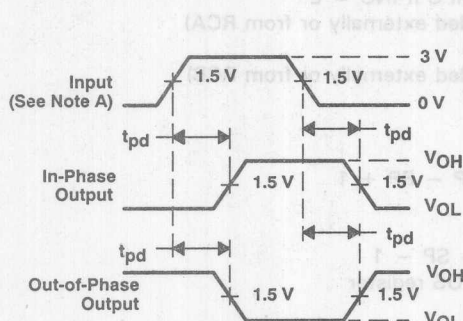
LOAD CIRCUIT



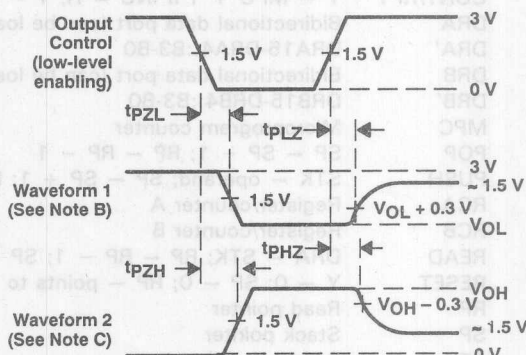
VOLTAGE WAVEFORMS
SETUP AND HOLD TIMES
INPUT RISE AND FALL TIMES



VOLTAGE WAVEFORMS
PULSE DURATION



VOLTAGE WAVEFORMS
PROPAGATION DELAY TIMES



VOLTAGE WAVEFORMS
ENABLE AND DISABLE TIMES, 3-STATE OUTPUTS

- Notes: A. Phase relationships between waveforms were chosen arbitrarily. All input pulses are supplied by pulse generators having the following characteristics: PRR = 1 MHz, Z_O = 50 Ω, t_r ≤ 6 ns, t_f ≤ 6 ns.
B. Waveform 1 is for an output with internal conditions such that the output is low except when disabled by the output control.
C. Waveform 2 is for an output with internal conditions such that the output is high except when disabled by the output control. For t_{PLZ} and t_{PHZ}, V_{OL} and V_{OH} are specified values.

FIGURE 1

PROGRAMMING INFORMATION

microprogramming the 'ACT8818A

Microprogramming is unlike programming monolithic processors for several reasons. First, the width of the microinstruction word is only partially constrained by the basic signals required to control the sequencer. Since the main advantage of a microprogrammed processor is speed, many operations are often supported by or carried out in special purpose hardware. Lookup tables, extra registers, address generators, elastic memories, and data acquisition circuits may also be controlled by the microinstruction.

The number of slices in a bit-slice ALU is user-defined, which makes the microinstruction width even more application dependent. Types of instructions resulting from manipulation of the sequencer controls are discussed below. Examples of some commonly used instructions can be found in the later section of microinstructions and flow diagrams. The following abbreviations are used in the tables in this section:

| | |
|------------|--|
| BR A | Y ← DRA |
| BR A' | Y ← DRA' |
| BR B | Y ← DRB |
| BR B' | Y ← DRB' |
| BR S | Y ← STK |
| CALL A | Y ← DRA; STK ← MPC; SP ← SP + 1; RP ← RP + 1 |
| CALL B | Y ← DRB; STK ← MPC; SP ← SP + 1; RP ← RP + 1 |
| CALL A' | Y ← DRA'; STK ← MPC; SP ← SP + 1; RP ← RP + 1 |
| CALL B' | Y ← DRB'; STK ← MPC; SP ← SP + 1; RP ← RP + 1 |
| CALL S | Y ← STK; STK ← MPC; SP ← SP + 1; RP ← RP + 1 |
| CLR SP, RP | SP ← 0; RP ← points to TOS register |
| CONT/RPT | Y ← MPC + 1 if INC = H; Y ← MPC if INC = L |
| DRA | Bidirectional data port (can be loaded externally or from RCA) |
| DRA' | DRA15-DRA4::B3-B0 |
| DRB | Bidirectional data port (can be loaded externally or from RCB) |
| DRB' | DRB15-DRB4::B3-B0 |
| MPC | Microprogram counter |
| POP | SP ← SP - 1; RP ← RP - 1 |
| PUSH | STK ← operand; SP ← SP + 1; RP ← RP + 1 |
| RCA | Register/counter A |
| RCB | Register/counter B |
| READ | DRA ← STK; RP ← RP - 1; SP ← SP - 1 |
| RESET | Y ← 0; SP ← 0; RP ← points to TOS register |
| RP | Read pointer |
| SP | Stack pointer |
| STK | Stack |

address selection

Y-output multiplexer controls MUX2-MUX0 select one of eight 3-source branches as shown in Table 2. The states of CC and ZERO determine which of the three sources is selected as the next address. ZERO is set at the beginning of any cycle in which a register/counter will decrement to zero. This applies to both internal ZERO and external ZEROUT signals.

PROGRAMMING INFORMATION

Table 2. Output Controls (MUX2-MUX0)

| MUX2-MUX0 | RESET | Y OUTPUT SOURCE | | |
|-----------|-------|---------------------|----------|---------------------|
| | | $\overline{CC} = L$ | | $\overline{CC} = H$ |
| | | ZERO = L | ZERO = H | |
| XXX | Yes | All Low | All Low | All Low |
| LLL | No | STK | MPC | DRA |
| LLH | No | STK | MPC | DRB |
| LHL | No | STK | DRA | MPC |
| LHH | No | STK | DRB | MPC |
| HLL | No | DRA | MPC | DRB |
| HLH | No | DRA [†] | MPC | DRB [‡] |
| HHL | No | DRA | STK | MPC |
| HHH | No | DRB | STK | MPC |

[†]DRA15-DRA4::B3-B0

[‡]DRB15-DRB4::B3-B0

By programming \overline{CC} high or low without decrementing registers, only one outcome is possible; thus, unconditional branches or continues can be implemented by forcing the condition code. Alternatively, \overline{CC} can be selected from status, in which case Branch A on Condition Code Else Branch B instructions are possible, where A and B are the address sources determined by MUX2-MUX0.

Decrement and Branch on Nonzero instructions, creating loops that repeat until a terminal count is reached, can be implemented by programming \overline{CC} low and decrementing a register/counter. If \overline{CC} is selected from status and registers are decremented, more complex instructions such as Exit on Condition Code or End or Loop are possible.

When MUX2-MUX0 = HLH, the B3-B0 inputs can replace the four least significant bits of DRA or DRB to create 16-way branches or, when \overline{CC} is based on status, to create 32-way branches.

stack controls

As in the case of the MUX controls, each stack-control coding is a three-way choice based on \overline{CC} and ZERO (see Table 3). This allows push, pop, or hold stack operations to occur in parallel with the aforementioned branches. A subroutine call is accomplished by combining a branch and push, while returns result from coding a branch to stack with a pop.

Table 3. Stack Controls (S2-S0)

| S2-S0 | OSEL | STACK OPERATION | | |
|-------|------|---------------------|-------------|---------------------|
| | | $\overline{CC} = L$ | | $\overline{CC} = H$ |
| | | ZERO = L | ZERO = H | |
| LLL | X | Reset/Clear | Reset/Clear | Reset/Clear |
| LLH | X | Clear SP/RP | Hold | Hold |
| LHL | X | Hold | Pop | Pop |
| LHH | X | Pop | Hold | Hold |
| HLL | X | Hold | Push | Push |
| HLH | X | Push | Hold | Hold |
| HHL | X | Push | Hold | Push |
| HHH | H | Read | Read | Read |
| HHH | L | Hold | Hold | Hold |



PROGRAMMING INFORMATION

Combining stack and MUX controls with status results and register decrements permits even greater complexity. For example: Return on Condition Code or End of Loop; Call A on Condition Code Else Branch to B; Decrement and Return on Nonzero; Call 16-Way.

Diagnostic stack dumps are possible using Read (S2-S0 = HHH) when OSEL is set high.

register controls

Unlike stack and MUX controls, register control is not dependent upon \overline{CC} and ZERO. Registers can be independently loaded, decremented, or held using register control inputs RC2-RC0 (see Table 4). All combinations are supported with the exception of simultaneous register decrements. The register control inputs can be set to store branch addresses and loop counts or to decrement loop counts, facilitating the complex branching instructions described above.

Table 4. Register Controls (RC2-RC0)

| RC2-RC0 | REGISTER OPERATIONS | |
|---------|---------------------|-----------|
| | REG A | REG B |
| LLL | Hold | Hold |
| LLH | Decrement | Hold |
| LHL | Load | Hold |
| LHH | Decrement | Load |
| HLL | Load | Load |
| HLH | Hold | Decrement |
| HHL | Hold | Load |
| HHH | Load | Decrement |

The contents of RCA are accessible to the DRA port when OSEL is low and the output bus is enabled by \overline{RAOE} being low. Data from RCB is available when DRB is enabled by \overline{RBOE} being low.

continue/repeat instructions

The most commonly used instruction is a continue, implemented by selecting MPC at the Y output MUX and setting INC high. If MPC is selected and INC is off, the current instruction will simply be repeated.

A repeat instruction can be implemented in two ways. A programmed repeat (INC forced low) may be useful in generating wait states, for example, wait for interrupt. A conditional repeat (INC originates from status) may be useful in implementing Do While operations. Several bit patterns in the MUX control field of the microinstruction will place MPC on the microaddress bus.

branch instructions

A branch or jump to a given microaddress can also be coded several ways. RCA, DRA, RCB, DRB, and STK are possible sources for branch addresses (see Table 2). Branches to register or stack are useful whenever the branch address could be stored to reduce overhead.

The simplest branches are to DRA and DRB, since they require only one cycle and the branch address is supplied in the microinstruction. Use of registers or stack requires an initial load cycle (which may be combined with a preceding instruction), but may be more practical when an entry point is referenced over and over throughout the microprogram, for example, in error-handling routines. Branches to stack or register also enhance sequencing techniques in which a branch address is dynamically computed or multiple branches to a common entry point are used, but the entry point varies according to the system state. In this case, the state change might require reloading the stack or register.

PROGRAMMING INFORMATION

In order to force a branch to DRA or DRB, \overline{CC} must be programmed high or low. A branch to stack is only possible when \overline{CC} is forced low (see Table 2).

When \overline{CC} is low, the ZERO flag is tested, and if a register decrements to zero, the branch will be transformed into a Decrement and Branch on Nonzero instruction. Therefore, registers should not be decremented during branch instructions using $\overline{CC} = 0$ unless it is certain the register will not reach terminal count. Call (Branch and Push MPC) instructions and Return (Branch to Stack and Pop) instructions are discussed in later sections.

conditional branch instructions

Perhaps the most useful of all branches is the conditional branch. The 'ACT8818A permits three modes of conditional branching: Branch on Condition Code; Branch 16-Way from DRA or DRB; and Branch on Condition Code 16-Way from DRA Else Branch 16-Way from DRB. This increases the versatility of the system and the speed of processing status tests because both single-bit and 4-bit status are allowed.

Testing single-bit status is preferred when the status can be set up and selected through a status MUX prior to the conditional branch. Four-bit status allows the 'ACT8818A to process instructions based on Boolean status expressions, such as Branch if Overflow and Not Carry if Zero or if Negative. It also permits true n-way branches, such as If Negative then Branch to X, Else if Overflow, and Not Carry then Branch to Y. The tradeoff is speed versus program size. Since multiway branching occurs relatively infrequently in most programs, users will enjoy increased speed at a negligible cost. Call (Branch and Push MPC) instructions and Return (Branch to Stack and Pop) instructions are discussed in later sections.

loop instructions

Up to two levels of nested loops are possible when both counters are used simultaneously. Loop count and levels of nesting can be increased by adding external counters if desired. The simplest and most widely used of the loop instructions is Decrement and Branch on Nonzero, in which \overline{CC} is forced low while a register is decremented. As before, many forms are possible, since the top-of-loop address can originate from RCA, DRA, RCB, DRB, or the stack (see Table 2). Upon terminal count, instruction flow can either drop out of the bottom of the loop or branch elsewhere.

When loops are used in conjunction with \overline{CC} as status, B3-B0 as status and/or stack manipulation, many useful instructions are possible, including Decrement and Branch on Nonzero else Return, Decrement and Call on Nonzero, and Decrement and Branch 16-Way on Nonzero. Possible variations are summarized in Table 5. Call (Branch and Push MPC) instructions and Return (Branch to Stack and Pop) instructions are discussed in later sections.

Another level of complexity is possible if \overline{CC} is selected from status while looping. This type of loop will exit either because \overline{CC} is true or because a terminal count has been reached. This makes it possible, for example, to search the ALU for a bit string. If the string is found, the match forces \overline{CC} high. However, if no match is found, it is necessary to terminate the process when the entire word has been scanned. This complex process can then be implemented in a simple compact loop using Conditional Decrement and Branch on Nonzero.

SN74ACT8818A
16-BIT MICROSEQUENCER

PROGRAMMING INFORMATION

Table 5. Decrement and Branch on Nonzero Encodings

| MUX2- MUX0 | SE-S0 | OSEL | CC = L | | CC = H |
|---------------|-------|------|---------------------------|----------------|----------------------|
| | | | ZERO = L | ZERO = H | |
| LLL | LLH | X | BR S: CLR SP/RP | CONT/RPT | BR A |
| LLL | LHL | X | BR S | CONT/RPT: POP | BR A: POP |
| LLL | HLL | X | BR S | CONT/RPT: PUSH | CALL A |
| LLL | HHH | 0 | BR S | CONT/RPT | BR A |
| LLL | HHH | 1 | BR S: READ | CONT/RPT: READ | BR A: READ |
| LLH | LLH | X | BR S: CLR SP/RP | CONT/RPT | BR B |
| LLH | LHL | X | BR S | CONT/RPT: POP | BR B: POP |
| LLH | HLL | X | BR S | CONT/RPT: PUSH | CALL B |
| LLH | HHH | 0 | BR S | CONT/RPT | BR B |
| LLH | HHH | 1 | BR S: READ | CONT/RPT: READ | BR B: READ |
| LHL | LLH | X | BR S: CLR SP/RP | BR A | CONT/RPT |
| LHL | LHL | X | BR S | BR A: POP | CONT/RPT: POP |
| LHL | HLL | X | BR S | CALL A | CONT/RPT: PUSH |
| LHL | HHH | 0 | BR S | BR A | CONT/RPT |
| LHL | HHH | 1 | BR S: READ | BR A: READ | CONT/RPT: READ |
| LHH | LLH | X | BR S: CLR SP/RP | BR B | CONT/RPT |
| LHH | LHL | X | BR S | BR B: POP | CONT/RPT: POP |
| LHH | HLL | X | BR S | CALL B | CONT/RPT: PUSH |
| LHH | HHH | 0 | BR S | BR B | CONT/RPT |
| LHH | HHH | 1 | BR S: READ | BR B: READ | CONT/RPT: READ |
| HLL | LLH | X | BR A: CLR SP/RP | CONT/RPT | BR B |
| HLL | LHL | X | BR A | CONT/RPT: POP | BR B: POP |
| HLL | LHH | X | BR A: POP | CONT/RPT | BR B |
| HLL | HLL | X | BR A | CONT/RPT: PUSH | CALL B |
| HLL | HHH | 0 | BR A | CONT/RPT | BR B |
| HLL | HHH | 1 | BR A: READ | CONT/RPT: READ | BR B: READ |
| HLH | LLH | X | BR A' (16-way): CLR SP/RP | CONT/RPT | BR B' (16-way) |
| HLH | LHL | X | BR A' (16-way) | CONT/RPT: POP | BR B' (16-way): POP |
| HLH | LHH | X | BR A' (16-way): POP | CONT/RPT | BR B' (16-way) |
| HLH | HLL | X | BR A' (16-way) | CONT/RPT: PUSH | CALL B' (16-way) |
| HLH | HHH | 0 | BR A' (16-way) | CONT/RPT | BR B' (16-way) |
| HLH | HHH | 1 | BR A' (16-way): READ | CONT/RPT: READ | BR B' (16-way): READ |
| HHL | LLH | X | BR A: CLR SP/RP | BR S | CONT/RPT |
| HHL | LHL | X | BR A | RET (BRS: POP) | CONT/RPT: POP |
| HHL | LHH | X | BR A: POP | BR S | CONT/RPT |
| HHL | HLL | X | BR A | CALL S | CONT/RPT: PUSH |
| HHL | HHH | 0 | BR A | BR S | CONT/RPT |
| HHL | HHH | 1 | BR A: READ | BR S: READ | CONT/RPT: READ |
| HHH | LLH | X | BR B: CLR SP/RP | BR S | CONT/RPT |
| HHH | LHL | X | BR B | RET | CONT/RPT: POP |
| HHH | LHH | X | BR B: POP | BR S | CONT/RPT |
| HHH | HLL | X | BR B | CALL S | CONT/RPT: PUSH |
| HHH | HHH | 0 | BR B | BR S | CONT/RPT |
| HHH | HHH | 1 | BR B: READ | BR S: READ | CONT/RPT: READ |



POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

PROGRAMMING INFORMATION

subroutine calls

The various branch instructions described above can be merged with a push instruction to implement subroutine calls in a single cycle. Calls, conditional calls, and Decrement and Call on Nonzero are the most obvious.

Since a push is conditional on \overline{CC} and ZERO, many hybrid instructions are also possible, such as Call X on Condition Code Else Branch, or Decrement and Return on Nonzero Else Branch. Codes that cause subroutine calls are summarized in Tables 6 and 7.

Table 6. Call Encodings without Register Decrements

| MUX2-MUX0 | S2-S0 | OSEL | $\overline{CC} = L$ (ZERO = L) | $\overline{CC} = H$ |
|-----------|-------|------|--------------------------------|---------------------|
| LLL | HLH | X | CALL S | BR A |
| LLL | HHL | X | CALL S | CALL A |
| LLH | HLH | X | CALL S | BR B |
| LLH | HHL | X | CALL S | CALL B |
| LHL | HLH | X | CALL S | CONT/RPT |
| LHL | HHL | X | CALL S | CONT/RPT: PUSH |
| LHH | HLH | X | CALL S | CONT/RPT |
| LHH | HHL | X | CALL S | CONT/RPT: PUSH |
| HLL | HLH | X | CALL A | BR B |
| HLL | HHL | X | CALL A | CALL B |
| HLH | HLH | X | CALL A' (16-way) | BR B' (16-way) |
| HLH | HHL | X | CALL A' (16-way) | CALL B' (16-way) |
| HHL | HLH | X | CALL A | CONT/RPT |
| HHL | HHL | X | CALL A | CONT/RPT: PUSH |
| HHH | HLH | X | CALL B | CONT/RPT |
| HHH | HHL | X | CALL B | CONT/RPT: PUSH |

Table 7. Call Encodings with Register Decrements

| MUX2-MUX0 | S2-S0 | OSEL | $\overline{CC} = L$ | | $\overline{CC} = H$ |
|-----------|-------|------|---------------------|----------|---------------------|
| | | | ZERO = L | ZERO = H | |
| LLL | HLH | X | CALL S | CONT/RPT | BR A |
| LLL | HHL | X | CALL S | CONT/RPT | CALL A |
| LLH | HLH | X | CALL S | CONT/RPT | BR B |
| LLH | HHL | X | CALL S | CONT/RPT | CALL B |
| LHL | HLH | X | CALL S | BR A | CONT/RPT |
| LHL | HHL | X | CALL S | BR A | CONT/RPT: PUSH |
| LHH | HLH | X | CALL S | BR B | CONT/RPT |
| LHH | HHL | X | CALL S | BR B | CONT/RPT: PUSH |
| HLL | HLH | X | CALL A | CONT/RPT | BR B |
| HLL | HHL | X | CALL A | CONT/RPT | CALL B |
| HLH | HLH | X | CALL A' (16-way) | CONT/RPT | BR B' (16-way) |
| HLH | HHL | X | CALL A' (16-way) | CONT/RPT | CALL B' (16-way) |
| HHL | HLH | X | CALL A | BR S | CONT/RPT |
| HHL | HHL | X | CALL A | BR S | CONT/RPT: PUSH |
| HHH | HLH | X | CALL B | BR S | CONT/RPT |
| HHH | HHL | X | CALL B | BR S | CONT/RPT: PUSH |

PROGRAMMING INFORMATION

subroutine returns

A return from subroutine can be implemented by coding a branch to stack with a pop. Since pop is also conditional on \overline{CC} and ZERO, the complex forms discussed previously also apply to return instructions: Decrement and Return on Nonzero; Return on Condition Code; Branch on Condition Code Else Return. Return encodings are summarized in Tables 8 and 9.

Table 8. Return Encodings without Register Decrements

| MUX2-MUX0 | S2-S0 | OSEL | $\overline{CC} = L$ | $\overline{CC} = H$ |
|-----------|-------|------|---------------------|---------------------|
| LLL | LHH | X | RET | BR A |
| LLH | LHH | X | RET | BR B |
| LHL | LHH | X | RET | CONT/RPT |
| LHH | LHH | X | RET | CONT/RPT |

Table 9. Return Encodings with Register Decrements

| MUX2-MUX0 | S2-S0 | OSEL | $\overline{CC} = L$ | | $\overline{CC} = H$ |
|-----------|-------|------|---------------------|----------|---------------------|
| | | | ZERO = L | ZERO = H | |
| LLL | LHH | X | RET | CONT/RPT | BR A |
| LLH | LHH | X | RET | CONT/RPT | BR B |
| LHL | LHH | X | RET | BR A | CONT/RPT |
| LHH | LHH | X | RET | BR B | CONT/RPT |
| HHL | LHL | X | BR A | RET | CONT/RPT: POP |
| HHH | LHL | X | BR B | RET | CONT/RPT: POP |

reset

Pulling the S2-S0 pins low clears the stack and read pointers, and zeroes the Y output multiplexer (See Table 3).

clear pointers

The stack and read pointers can be cleared without affecting the Y output multiplexer by setting S2-S0 to LLH and forcing \overline{CC} low (see Table 3).

read stack

Placing a high value on all of the stack inputs (S2-S0) and OSEL places the 'ACT8818A into the read mode. At each low-to-high clock transition, the address pointed to by the read pointer is available at the DRA port and the read pointer is decremented. The bottom of the stack is detected by monitoring the stack warning/read error pin (STKW RN/RER). A high appears on the STKW RN/RER output when the stack contains one word and a read instruction is applied to the S2-S0 pins. This signifies that the last address has been read.

The stack pointer and stack contents are unaffected by the read operation. Under normal push and pop operations, the read pointer is updated with the stack pointer and contains identical information.

PROGRAMMING INFORMATION

interrupts

Real-time vectored interrupt routines are supported for those applications where polling would impede system throughput. Any instruction, including pushes and pops, may be interrupted. To process an interrupt, the following procedure should be followed:

1. Place the bidirectional Y bus into a high-impedance state by forcing \overline{YOE} high.
2. Force the interrupt entry point vector onto the Y bus. INC should be high.
3. Push the current value in the Interrupt Return register on the stack as the execution address to return to when interrupt handling is complete.

The first instruction of the interrupt routine must push the address stored in the interrupt return register onto the stack so that proper return linkage is maintained. This is accomplished by setting \overline{INT} and B1 low and coding a push on the stack.

sample microinstructions for the 'ACT8818A

Representative examples of instructions using the 'ACT8818A are given below. The examples assume a one-level pipeline system, in which the address and contents of the next instruction are being fetched while the current instruction is being executed, and an ALU status register contains the status results of the previous instruction.

Since the incrementer looks two addresses ahead of the address in the instruction register to set up some instructions such as continue or repeat, a set-up instruction has been included with each example. This shows the required state of both INC and \overline{CC} . \overline{CC} must be set up early because the status register on which Y-output selection is typically based contains the results of the previous instruction.

Flow diagrams and suggested code for the sample microinstructions are also given below. Numbers inside the circles are microword address locations expressed as hexadecimal numbers. Fields in microinstructions are binary numbers except for inputs on DRA or DRB, which are also in hexadecimal. For a discussion of sequencing instructions, see the preceding section on microprogramming.

continue

To continue (Instruction 10), INC and \overline{CC} must be programmed high one cycle ahead of instruction 10 for pipelining.

| Address | Instruction | MUX2-MUX0 | S2-S0 | R2-R0 | OSEL | \overline{CC} | INC | DRA | DRB |
|----------|-------------|-----------|-------|-------|------|-----------------|-----|------|------|
| (Set-up) | | XXX | XXX | XXX | X | 1 | 1 | XXXX | XXXX |
| 10 | Continue | 110 | 111 | XXX | 0 | X | X | XXXX | XXXX |

continue and pop

To continue and decrement the stack pointer (Pop), INC and \overline{CC} are forced high in the previous instruction.

| Address | Instruction | MUX2-MUX0 | S2-S0 | R2-R0 | OSEL | \overline{CC} | INC | DRA | DRB |
|----------|--------------|-----------|-------|-------|------|-----------------|-----|------|------|
| (Set-up) | | XXX | XXX | XXX | X | 1 | 1 | XXXX | XXXX |
| 10 | Continue/Pop | 110 | 010 | XXX | X | X | X | XXXX | XXXX |

PROGRAMMING INFORMATION

continue and push

To continue and push the microprogram counter onto the stack (Push), INC and \overline{CC} are forced high one cycle ahead of Instruction 10 for pipelining.

| Address | Instruction | MUX2-MUX0 | S2-S0 | R2-R0 | OSEL | \overline{CC} | INC | DRA | DRB |
|----------|---------------|-----------|-------|-------|------|-----------------|-----|------|------|
| (Set-up) | | XXX | XXX | XXX | X | 1 | 1 | XXXX | XXXX |
| 10 | Continue/Push | 110 | 100 | XXX | 0 | X | X | XXXX | XXXX |

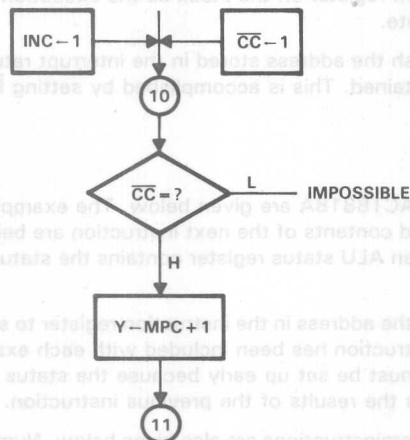


FIGURE 2. CONTINUE

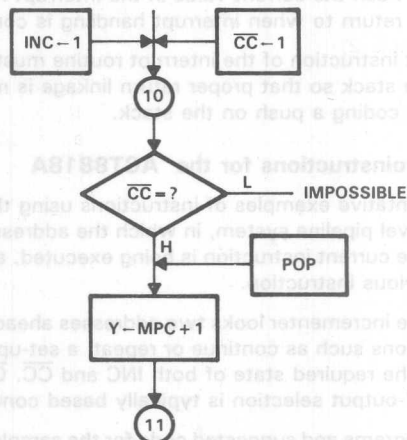


FIGURE 3. CONTINUE AND POP

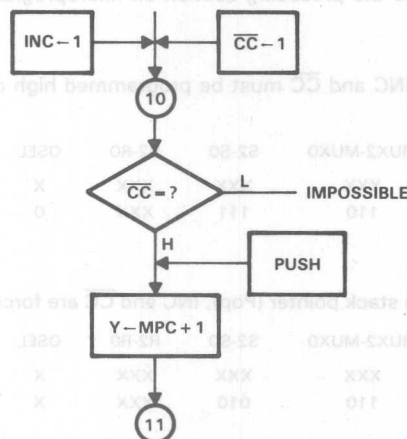


FIGURE 4. CONTINUE AND PUSH

PROGRAMMING INFORMATION

branch (example 1)

To branch from address 10 to address 20, \overline{CC} must be programmed high one cycle ahead of Instruction 10 for pipelining.

| Address | Instruction | MUX2-MUX0 | S2-S0 | R2-R0 | OSEL | \overline{CC} | INC | DRA | DRB |
|----------|-------------|-----------|-------|-------|------|-----------------|-----|------|------|
| (Set-up) | | XXX | XXX | XXX | X | 1 | X | XXXX | XXXX |
| 10 | BR A | 000 | 111 | XXX | 0 | X | X | 0020 | XXXX |

branch (example 2)

To branch from address 10 to address 20, \overline{CC} is programmed low in the previous instruction; as a result, a ZERO test follows the condition code test in Instruction 10. To ensure that a ZERO = H condition will not occur, registers should not be decremented during this instruction.

| Address | Instruction | MUX2-MUX0 | S2-S0 | R2-R0 | OSEL | \overline{CC} | INC | DRA | DRB |
|----------|-------------|-----------|-------|-------|------|-----------------|-----|------|------|
| (Set-up) | | XXX | XXX | XXX | X | 0 | X | XXXX | XXXX |
| 10 | BR A | 110 | 111 | 000 | 0 | X | X | 0020 | XXXX |

sixteen-way branch

To branch 16-way, \overline{CC} is programmed high in the previous instruction. The branch address is derived from the concatenation DRB15-DRB4::B3-B0.

| Address | Instruction | MUX2-MUX0 | S2-S0 | R2-R0 | OSEL | \overline{CC} | INC | DRA | DRB |
|----------|-------------|-----------|-------|-------|------|-----------------|-----|------|------|
| (Set-up) | | XXX | XXX | XXX | X | 1 | X | XXXX | XXXX |
| 10 | BR B' | 101 | 111 | XXX | 0 | X | X | XXXX | 0040 |

conditional branch

To branch to address 20 Else Continue to address 11, INC is set high in the preceding instruction to set up the Continue.

| Address | Instruction | MUX2-MUX0 | S2-S0 | R2-R0 | OSEL | CC' | INC | DRA | DRB |
|----------|-----------------------|-----------|-------|-------|------|-----|-----|------|------|
| (Set-up) | | XXX | XXX | XXX | X | X | 1 | XXXX | XXXX |
| 10 | BR A else Continue | 110 | 111 | 000 | 0 | X | X | 0020 | XXXX |

PROGRAMMING INFORMATION

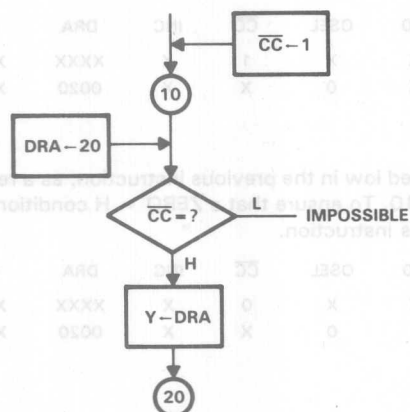


FIGURE 5. BRANCH EXAMPLE 1

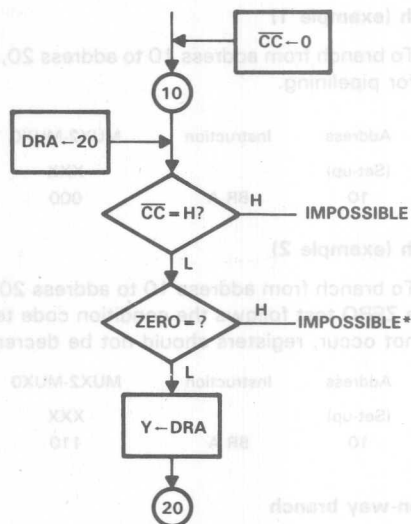


FIGURE 6. BRANCH EXAMPLE 2

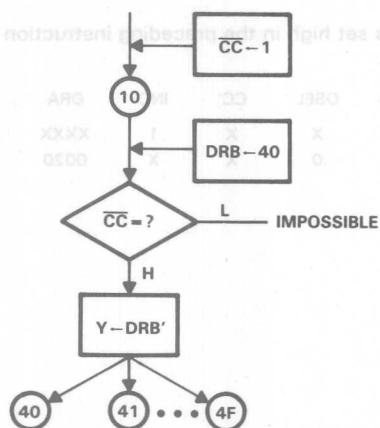


FIGURE 7. SIXTEEN-WAY BRANCH

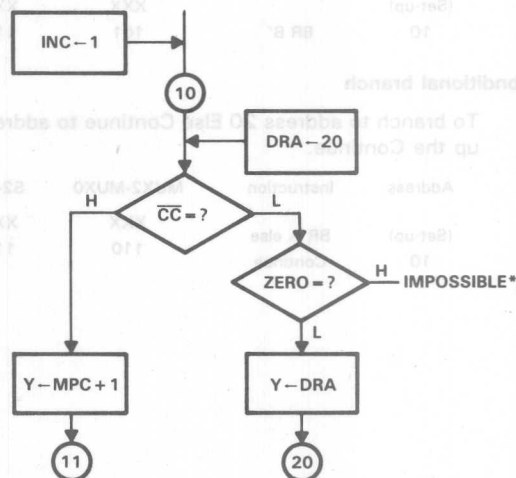


FIGURE 8. CONDITIONAL BRANCH

PROGRAMMING INFORMATION

three-way branch

To branch 3-way, this example uses an instruction from Table 5 with BR A in the ZERO = L column, CONT/RPT in the ZERO = H column and BR B in the CC = H column. To enable the ZERO = H path, register A must decrement to zero during this instruction (see Table 4 for possible register operations). INC is programmed high in Instruction 10 to set up the Continue.

| Address | Instruction | MUX2-MUX0 | S2-S0 | R2-R0 | OSEL | CC | INC | DRA | DRB |
|----------|----------------------------------|-----------|-------|-------|------|----|-----|------|------|
| (Set-up) | | XXX | XXX | XXX | X | 1 | 1 | XXXX | XXXX |
| 10 | Continue and Load Reg A | 110 | 111 | 010 | 0 | 1 | 1 | XXXX | XXXX |
| 11 | Decrement Reg A; Branch 3-way | 100 | 111 | 001 | 0 | X | X | 0020 | 0030 |

[†]Selected from external status

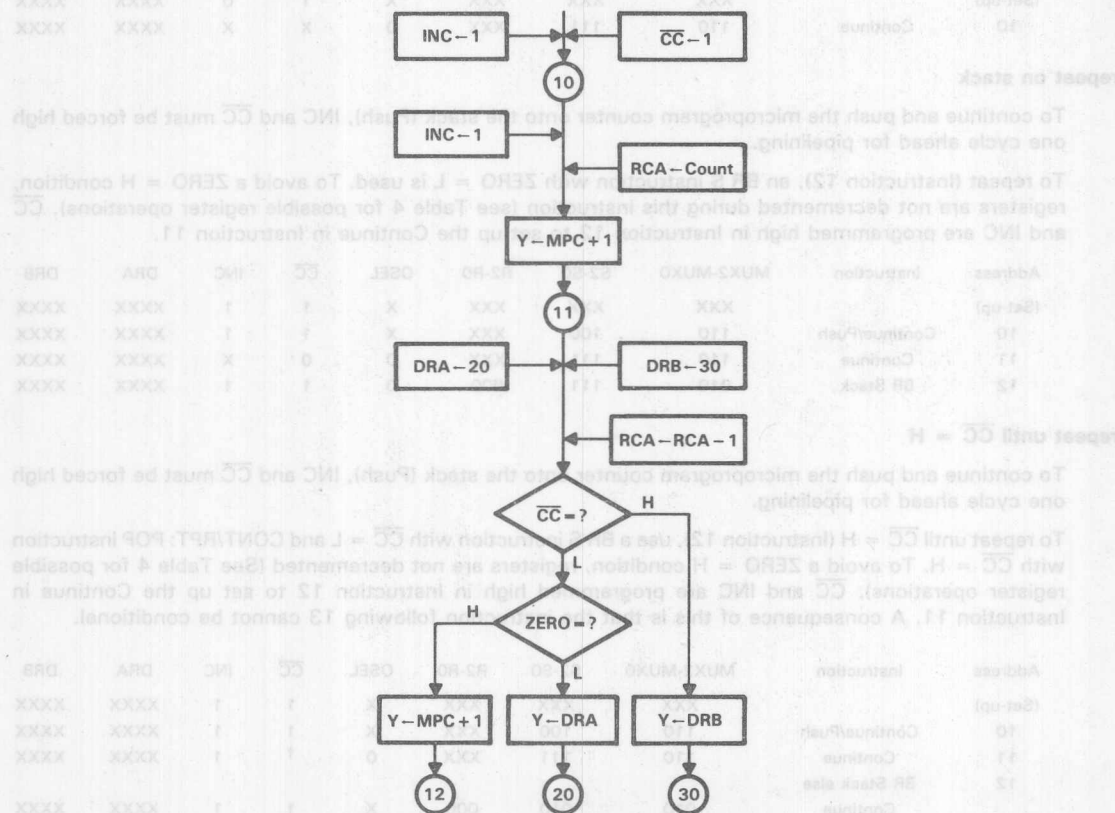


FIGURE 9. THREE-WAY BRANCH

PROGRAMMING INFORMATION

thirty-two-way branch

To branch 32-way, the four least significant bits of the DRA' and DRB' addresses must be input at the B3-B0 port; these are concatenated with the 12 most significant bits of DRA and DRB to provide new addresses DRA' (DRA15-DRA4::B3-B0) and DRB' (DRB15-DRB4::B3-B0).

| Address | Instruction | MUX2-MUX0 | S2-S0 | R2-R0 | OSEL | \overline{CC} | INC | DRA | DRB |
|----------|---------------|-----------|-------|-------|------|-----------------|-----|------|------|
| (Set-up) | | XXX | XXX | XXX | X | X | 1 | XXXX | XXXX |
| 10 | 32-way Branch | 101 | 111 | 000 | 0 | X | X | 0040 | 0030 |

repeat

To repeat (Instruction 10), INC must be programmed low and \overline{CC} high one cycle ahead of Instruction 10 for pipelining.

| Address | Instruction | MUX2-MUX0 | S2-S0 | R2-R0 | OSEL | \overline{CC} | INC | DRA | DRB |
|----------|-------------|-----------|-------|-------|------|-----------------|-----|------|------|
| (Set-up) | | XXX | XXX | XXX | X | 1 | 0 | XXXX | XXXX |
| 10 | Continue | 110 | 111 | XXX | 0 | X | X | XXXX | XXXX |

repeat on stack

To continue and push the microprogram counter onto the stack (Push), INC and \overline{CC} must be forced high one cycle ahead for pipelining.

To repeat (Instruction 12), an BR S instruction with ZERO = L is used. To avoid a ZERO = H condition, registers are not decremented during this instruction (see Table 4 for possible register operations). \overline{CC} and INC are programmed high in Instruction 12 to set up the Continue in Instruction 11.

| Address | Instruction | MUX2-MUX0 | S2-S0 | R2-R0 | OSEL | \overline{CC} | INC | DRA | DRB |
|----------|---------------|-----------|-------|-------|------|-----------------|-----|------|------|
| (Set-up) | | XXX | XXX | XXX | X | 1 | 1 | XXXX | XXXX |
| 10 | Continue/Push | 110 | 100 | XXX | X | 1 | 1 | XXXX | XXXX |
| 11 | Continue | 110 | 111 | XXX | 0 | 0 | X | XXXX | XXXX |
| 12 | BR Stack | 010 | 111 | 000 | 0 | 1 | 1 | XXXX | XXXX |

repeat until $\overline{CC} = H$

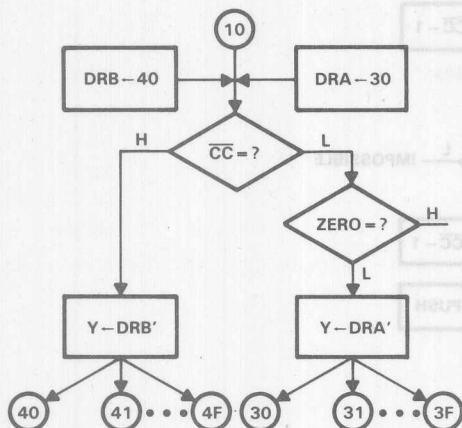
To continue and push the microprogram counter onto the stack (Push), INC and \overline{CC} must be forced high one cycle ahead for pipelining.

To repeat until $\overline{CC} = H$ (Instruction 12), use a BR S instruction with $\overline{CC} = L$ and CONT/RPT: POP instruction with $\overline{CC} = H$. To avoid a ZERO = H condition, registers are not decremented (See Table 4 for possible register operations). \overline{CC} and INC are programmed high in Instruction 12 to set up the Continue in Instruction 11. A consequence of this is that the instruction following 13 cannot be conditional.

| Address | Instruction | MUX2-MUX0 | S2-S0 | R2-R0 | OSEL | \overline{CC} | INC | DRA | DRB |
|----------|---------------------------|-----------|-------|-------|------|-----------------|-----|------|------|
| (Set-up) | | XXX | XXX | XXX | X | 1 | 1 | XXXX | XXXX |
| 10 | Continue/Push | 110 | 100 | XXX | X | 1 | 1 | XXXX | XXXX |
| 11 | Continue | 110 | 111 | XXX | 0 | † | 1 | XXXX | XXXX |
| 12 | BR Stack else Continue | 010 | 010 | 000 | X | 1 | 1 | XXXX | XXXX |

†Selected from external status

PROGRAMMING INFORMATION



*no register decrement

FIGURE 10. THIRTY-TWO-WAY BRANCH

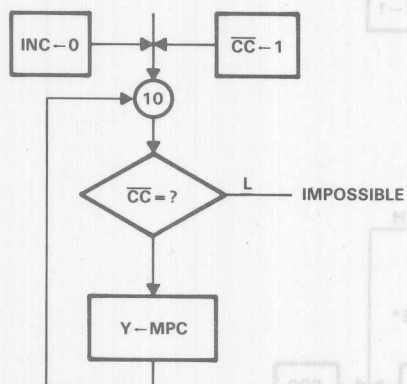
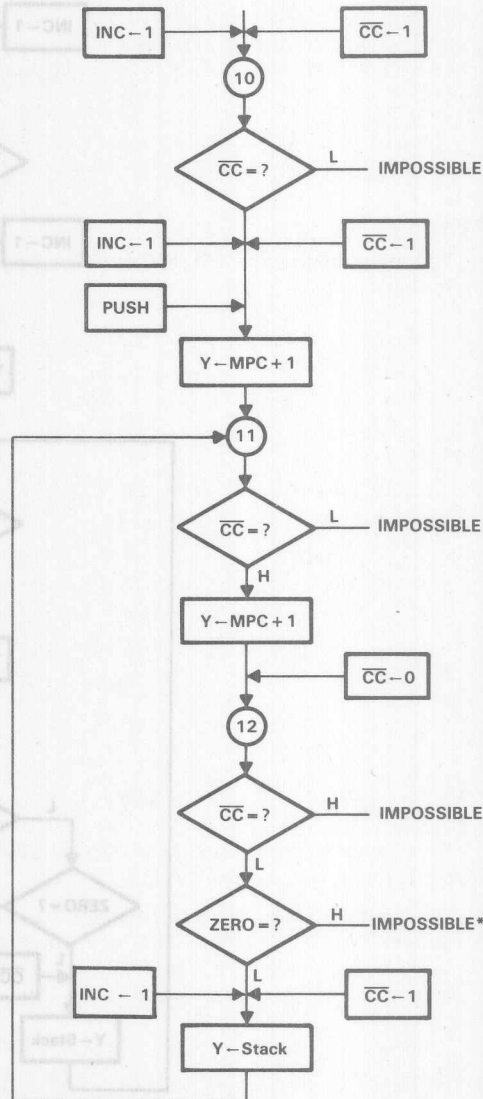


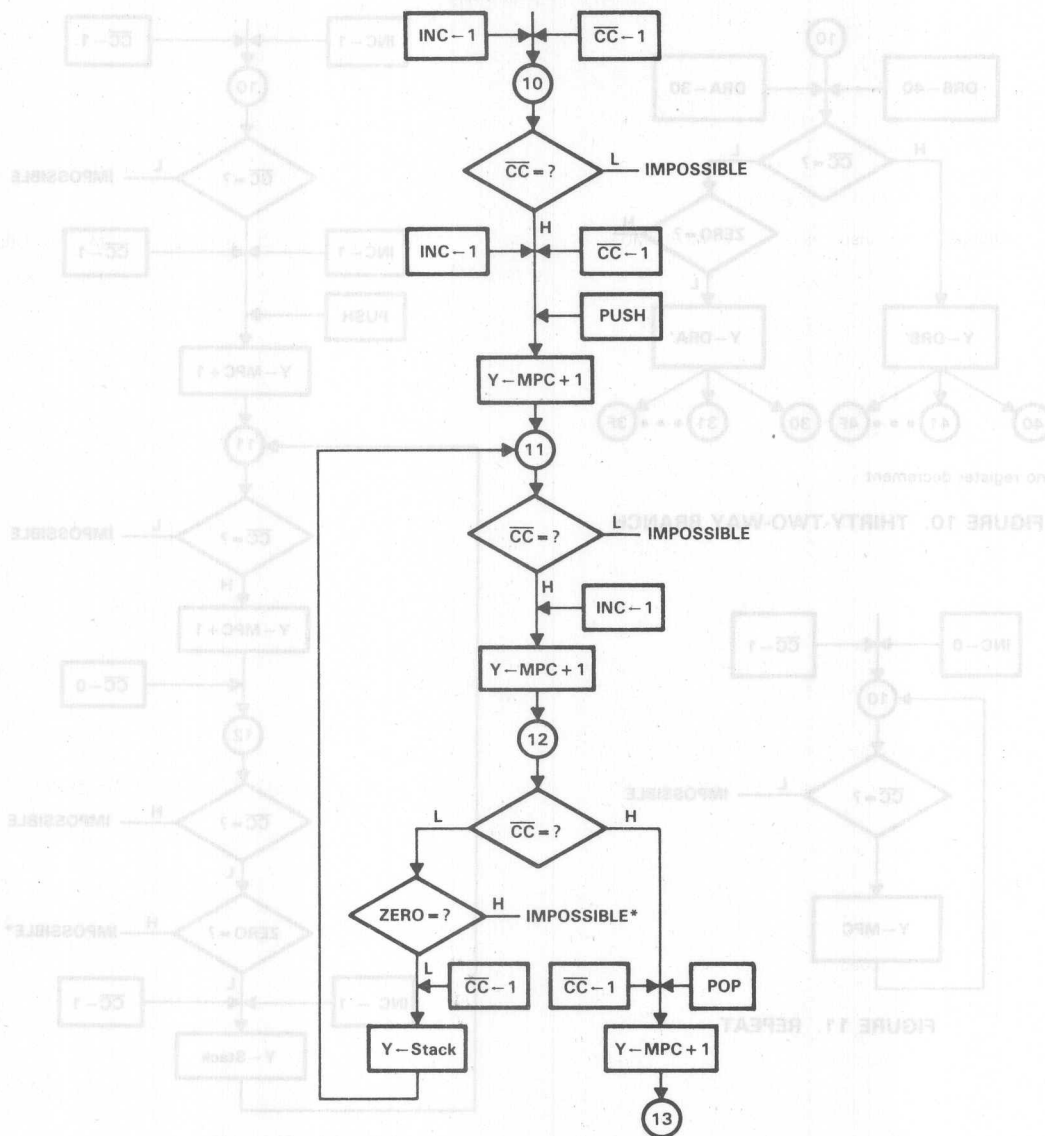
FIGURE 11. REPEAT



*no register decrement

FIGURE 12. REPEAT ON STACK

PROGRAMMING INFORMATION



*no register decrement

FIGURE 13. REPEAT UNTIL $\overline{CC} = H$

PROGRAMMING INFORMATION

loop until zero

To continue and push the microprogram counter onto the stack (Push), INC and \overline{CC} are forced high one cycle ahead for pipelining. Register A is loaded with the loop counter using a Load A instruction from Table 4.

To decrement the loop count, a decrement register A and hold register B instruction from Table 4 is used. To Repeat Else Continue and Pop (decrement the stack pointer), an instruction from Table 5 with BR S in the ZERO = L column and CONT/RPT: POP in the ZERO = H column is used. \overline{CC} is programmed low in Instruction 11 to force the ZERO test in Instruction 12; it is programmed high in Instruction 12 to set up the Continue in Instruction 11.

| Address | Instruction | MUX2-MUX0 | S2-S0 | R2-R0 | OSEL | \overline{CC} | INC | DRA | DRB |
|----------|--|-----------|-------|-------|------|-----------------|-----|------|------|
| (Set-up) | | XXX | XXX | XXX | X | 1 | 1 | XXXX | XXXX |
| 10 | Continue/Push | 110 | 100 | XXX | 0 | 1 | 1 | XXXX | XXXX |
| 11 | Continue/Load Reg A | 110 | 111 | 010 | 0 | 0 | 1 | XXXX | XXXX |
| 12 | Decrement Reg A; BR S else Continue: Pop | 000 | 010 | 001 | 1 | 1 | 1 | XXXX | XXXX |

conditional loop until zero

Two examples of a Conditional Loop on Stack with Exit are presented below. Both use the microcode shown below to branch to the stack on nonzero, continue and pop on zero, and branch to DRA with a pop if \overline{CC} = H. In the first example, the value on the DRA bus is the same as the value in the microprogram counter, making the exit destinations on the \overline{CC} and ZERO tests the same. In the second, the values are different, generating a two-way exit.

To continue and push the microprogram counter onto the stack (Push), INC must be high. \overline{CC} is forced high in the preceding instruction for pipelining.

To continue (Instruction 11), INC must be high. \overline{CC} must be programmed high in the previous instruction. INC is programmed high to set up the Continue in Instruction 12.

To decrement and branch else exit (Instruction 12), an instruction from Table 5 with BR S in the ZERO = L column, CONT/RPT: POP in the ZERO = H column and BR A: POP in the \overline{CC} = H column is used.

Example 1:

| Address | Instruction | MUX2-MUX0 | S2-S0 | R2-R0 | OSEL | \overline{CC} | INC | DRA | DRB |
|----------|--|-----------|-------|-------|------|-----------------|-----|------|------|
| (Set-up) | | XXX | XXX | XXX | X | 1 | 1 | XXXX | XXXX |
| 10 | Continue/Push Load Reg A | 110 | 100 | 010 | X | 1 | 1 | XXXX | XXXX |
| 11 | Continue | 110 | 111 | XXX | 0 | † | 1 | XXXX | XXXX |
| 12 | Decrement Reg A; BR S else Continue: Pop else BR A: Pop | 000 | 010 | 001 | X | 1 | 1 | 0013 | XXXX |

†Selected from external status

PROGRAMMING INFORMATION

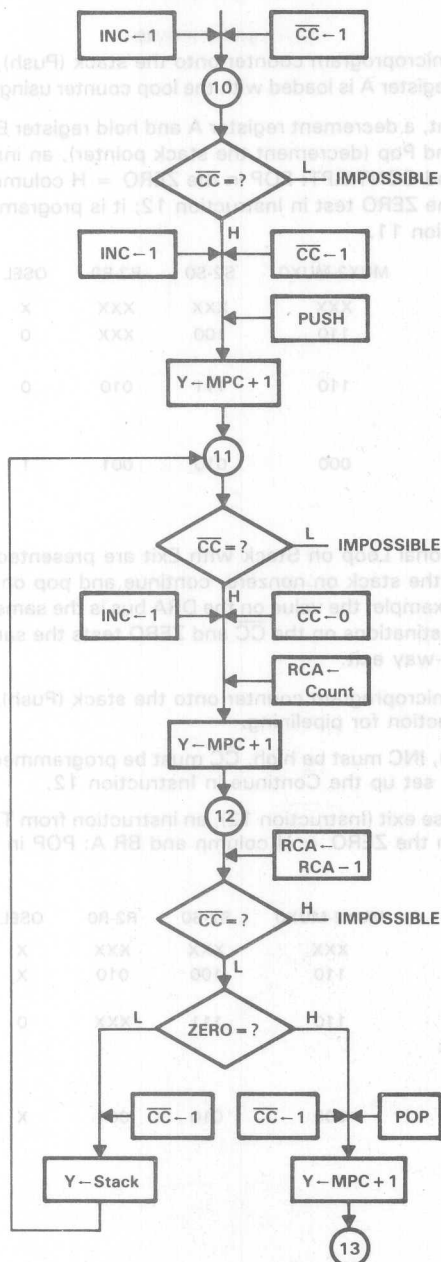


FIGURE 14. LOOP UNTIL ZERO

PROGRAMMING INFORMATION

Example 2:

| Address | Instruction | MUX2-MUX0 | S2-S0 | R2-R0 | OSEL | CC | INC | DRA | DRB |
|----------|--|-----------|-------|-------|------|----|-----|------|------|
| (Set-up) | | XXX | XXX | XXX | X | 1 | 1 | XXXX | XXXX |
| 10 | Continue/Push | 110 | 100 | 010 | X | 1 | 1 | XXXX | XXXX |
| 11 | Load Reg A | | | | | | | | |
| 12 | Continue | 110 | 111 | XXX | 0 | † | 1 | XXXX | XXXX |
| | Decrement Reg A; BR S else Continue: Pop else BR A: Pop | 000 | 010 | 001 | X | 1 | 1 | 0025 | XXXX |

†Selected from external status

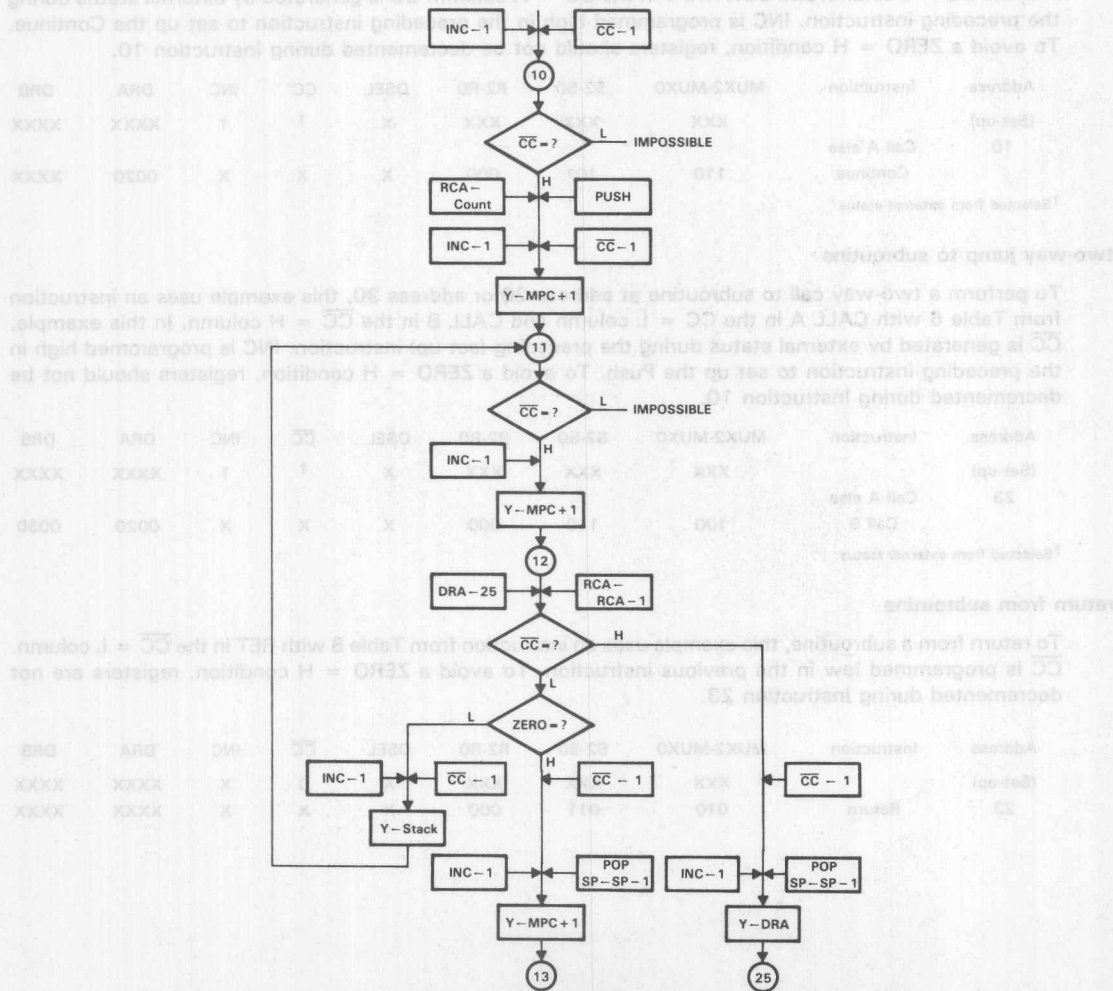


FIGURE 15. CONDITIONAL LOOP UNTIL ZERO (EXAMPLE 2)

PROGRAMMING INFORMATION

jump to subroutine

To call a subroutine at address 30, this example uses the instruction from Table 6 with CALL A in the $\overline{CC} = H$ column. \overline{CC} is programmed high in the previous instruction.

| Address | Instruction | MUX2-MUX0 | S2-S0 | R2-R0 | OSEL | \overline{CC} | INC | DRA | DRB |
|----------|-------------|-----------|-------|-------|------|-----------------|-----|------|------|
| (Set-up) | | XXX | XXX | XXX | X | 1 | 1 | XXXX | XXXX |
| 10 | Call A | 000 | 110 | XXX | X | X | X | 0030 | XXXX |

conditional jump to subroutine

To conditionally call a subroutine at address 20, this example uses an instruction from Table 6 with CALL A in the $\overline{CC} = L$ column and CONT/RPT in the $\overline{CC} = H$ column. \overline{CC} is generated by external status during the preceding instruction. INC is programmed high in the preceding instruction to set up the Continue. To avoid a ZERO = H condition, registers should not be decremented during Instruction 10.

| Address | Instruction | MUX2-MUX0 | S2-S0 | R2-R0 | OSEL | \overline{CC}' | INC | DRA | DRB |
|----------|-------------------------|-----------|-------|-------|------|------------------|-----|------|------|
| (Set-up) | | XXX | XXX | XXX | X | † | 1 | XXXX | XXXX |
| 10 | Call A else Continue | 110 | 101 | 000 | X | X | X | 0020 | XXXX |

†Selected from external status

two-way jump to subroutine

To perform a two-way call to subroutine at address 20 or address 30, this example uses an instruction from Table 6 with CALL A in the $\overline{CC} = L$ column and CALL B in the $\overline{CC} = H$ column. In this example, \overline{CC} is generated by external status during the preceding (set-up) instruction. INC is programmed high in the preceding instruction to set up the Push. To avoid a ZERO = H condition, registers should not be decremented during Instruction 10.

| Address | Instruction | MUX2-MUX0 | S2-S0 | R2-R0 | OSEL | \overline{CC} | INC | DRA | DRB |
|----------|-----------------------|-----------|-------|-------|------|-----------------|-----|------|------|
| (Set-up) | | XXX | XXX | XXX | X | † | 1 | XXXX | XXXX |
| 23 | Call A else Call B | 100 | 110 | 000 | X | X | X | 0020 | 0030 |

†Selected from external status

return from subroutine

To return from a subroutine, this example uses an instruction from Table 8 with RET in the $\overline{CC} = L$ column. \overline{CC} is programmed low in the previous instruction. To avoid a ZERO = H condition, registers are not decremented during Instruction 23.

| Address | Instruction | MUX2-MUX0 | S2-S0 | R2-R0 | OSEL | \overline{CC} | INC | DRA | DRB |
|----------|-------------|-----------|-------|-------|------|-----------------|-----|------|------|
| (Set-up) | | XXX | XXX | XXX | X | 0 | X | XXXX | XXXX |
| 23 | Return | 010 | 011 | 000 | X | X | X | XXXX | XXXX |

PROGRAMMING INFORMATION

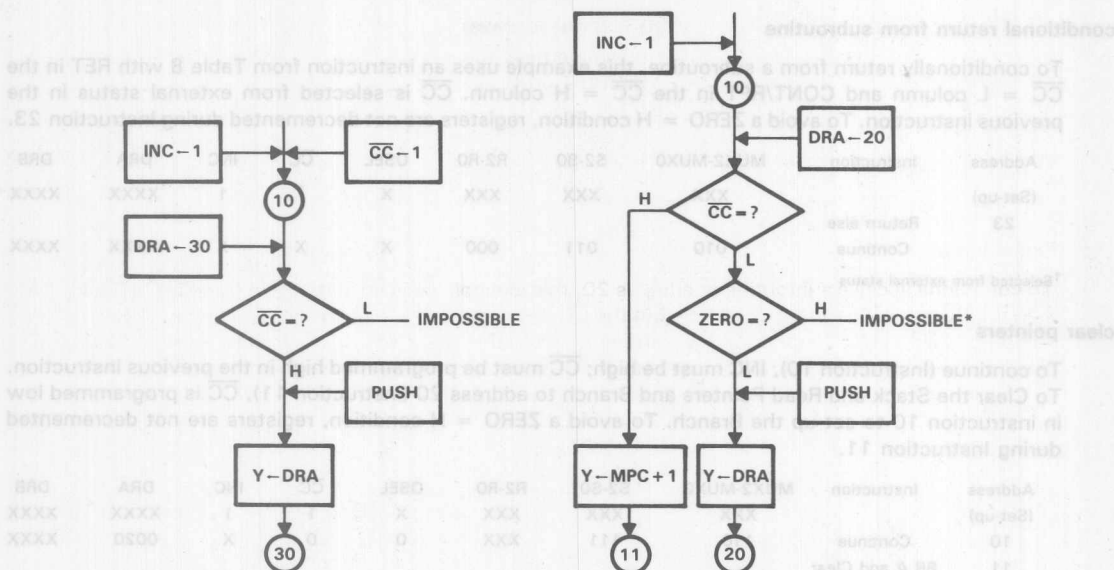
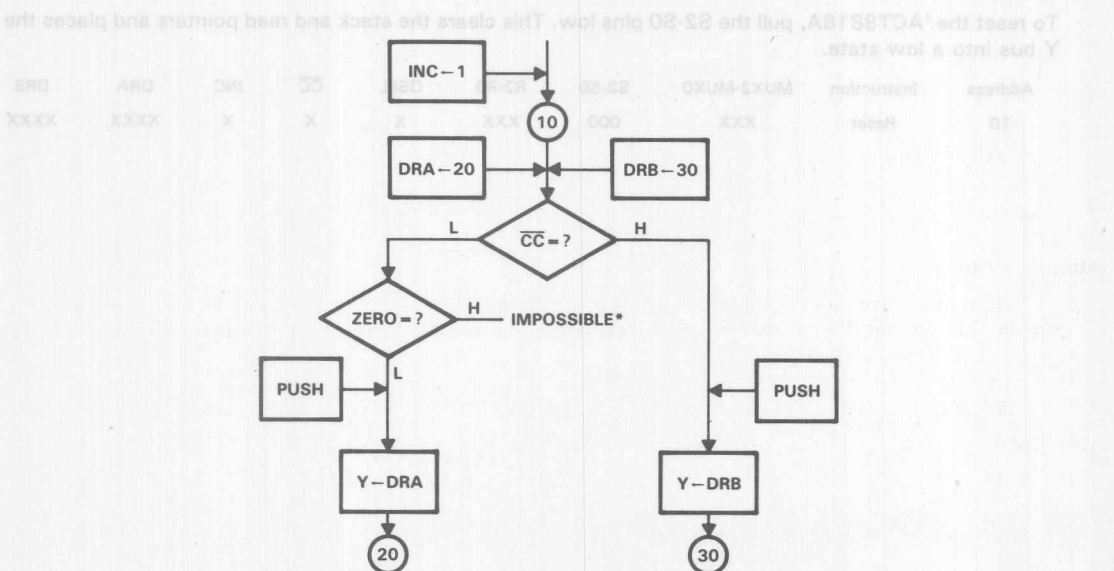


FIGURE 16. JUMP TO SUBROUTINE

*no register decrement

FIGURE 17. CONDITIONAL JUMP TO SUBROUTINE



*no register decrement

FIGURE 18. TWO-WAY JUMP TO SUBROUTINE

PROGRAMMING INFORMATION

conditional return from subroutine

To conditionally return from a subroutine, this example uses an instruction from Table 8 with RET in the \overline{CC} = L column and CONT/RPT in the \overline{CC} = H column. \overline{CC} is selected from external status in the previous instruction. To avoid a ZERO = H condition, registers are not decremented during Instruction 23.

| Address | Instruction | MUX2-MUX0 | S2-S0 | R2-R0 | OSEL | \overline{CC} | INC | DRA | DRB |
|----------|-------------------------|-----------|-------|-------|------|-----------------|-----|------|------|
| (Set-up) | | XXX | XXX | XXX | X | † | 1 | XXXX | XXXX |
| 23 | Return else Continue | 010 | 011 | 000 | X | X | X | XXXX | XXXX |

†Selected from external status

clear pointers

To continue (Instruction 10), INC must be high; \overline{CC} must be programmed high in the previous instruction. To Clear the Stack and Read Pointers and Branch to address 20 (instruction 11), \overline{CC} is programmed low in instruction 10 to set up the Branch. To avoid a ZERO = H condition, registers are not decremented during Instruction 11.

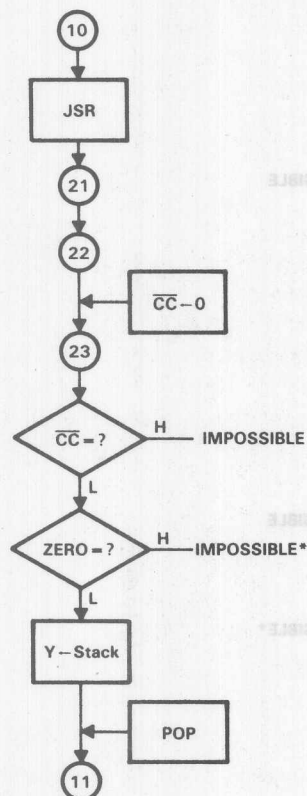
| Address | Instruction | MUX2-MUX0 | S2-S0 | R2-R0 | OSEL | \overline{CC} | INC | DRA | DRB |
|----------|-------------------------|-----------|-------|-------|------|-----------------|-----|------|------|
| (Set-up) | | XXX | XXX | XXX | X | 1 | 1 | XXXX | XXXX |
| 10 | Continue | 110 | 111 | XXX | 0 | 0 | X | 0020 | XXXX |
| 11 | BR A and Clear SP/RP | 110 | 001 | 000 | X | X | X | XXXX | XXXX |

reset

To reset the 'ACT8818A, pull the S2-S0 pins low. This clears the stack and read pointers and places the Y bus into a low state.

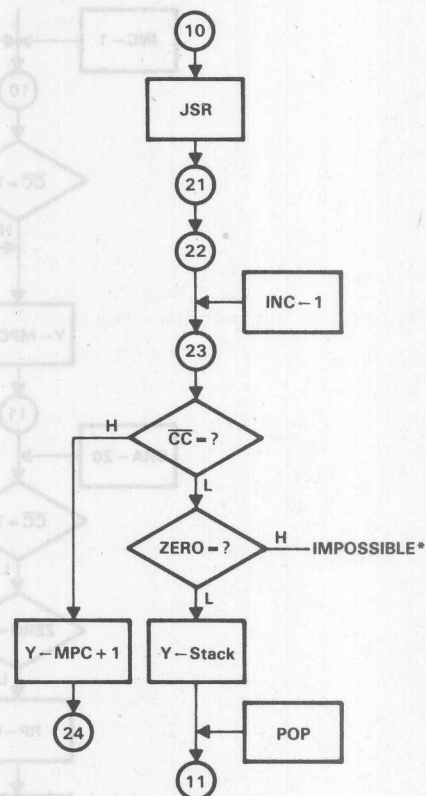
| Address | Instruction | MUX2-MUX0 | S2-S0 | R2-R0 | OSEL | \overline{CC} | INC | DRA | DRB |
|---------|-------------|-----------|-------|-------|------|-----------------|-----|------|------|
| 10 | Reset | XXX | 000 | XXX | X | X | X | XXXX | XXXX |

PROGRAMMING INFORMATION



*no register decrement

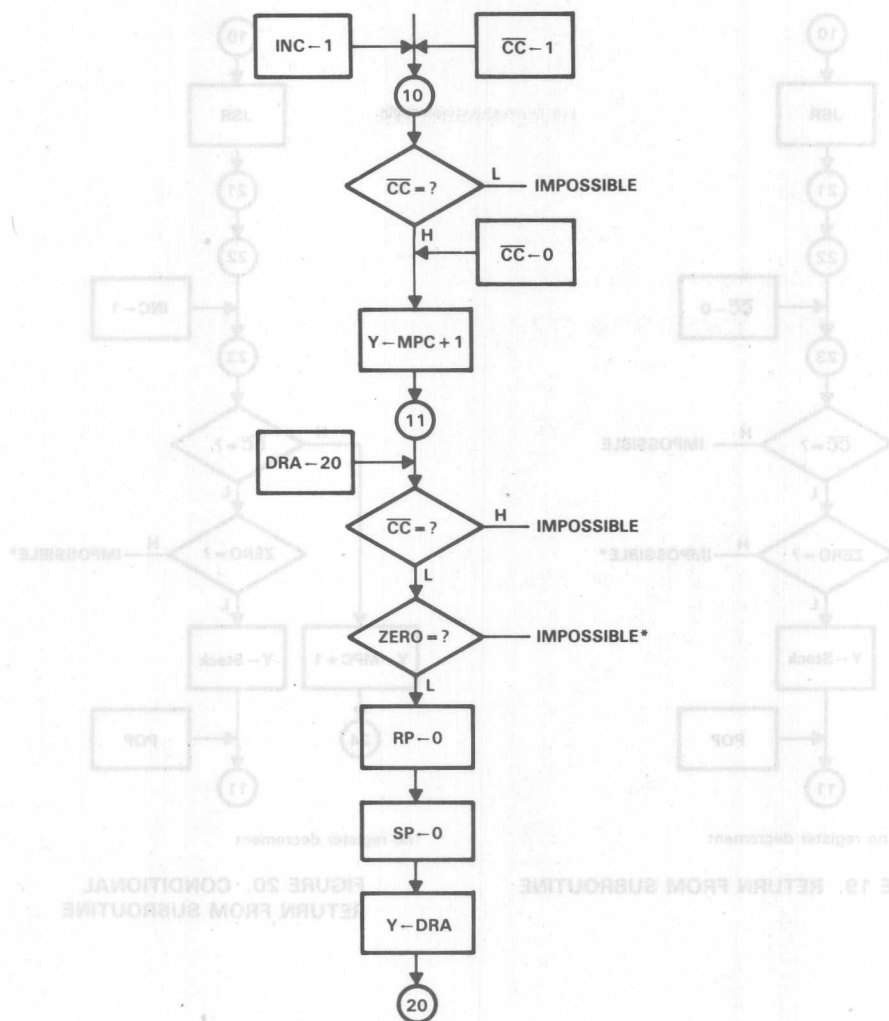
FIGURE 19. RETURN FROM SUBROUTINE



*no register decrement

FIGURE 20. CONDITIONAL RETURN FROM SUBROUTINE

PROGRAMMING INFORMATION



*no register decrement

FIGURE 21. CLEAR POINTERS

| | |
|---|-----------|
| General Information | 1 |
| SN74ACT8818A 16-Bit Microsequencer | 2 |
| SN74ACT8832A 32-Bit Registered ALU | 3 |
| SN74ACT8836A 32- × 32-Bit Parallel Multiplier | 4 |
| SN74ACT8841A Digital Crossbar Switch | 5 |
| SN74ACT8847 64-Bit Floating-Point/Integer Unit | 6 |
| SN74ACT8847 Application Information | 7 |
| SN74ACT8867 32-Bit Vector Processor Unit | 8 |
| Design Support | 9 |
| Mechanical Data | 10 |

| | |
|----|--|
| 1 | General Information |
| 2 | SN7A0CT8813A 16-Bit Microsequencer |
| 3 | SN7A0CT8832A 32-Bit Registered ALU |
| 4 | SN7A0CT8836A 32 × 32-Bit Parallel Multiplier |
| 5 | SN7A0CT8841A Digital Crossover Switch |
| 6 | SN7A0CT8847 64-Bit Floating-Point Integer Unit |
| 7 | SN7A0CT8847 Application Information |
| 8 | SN7A0CT8867 32-Bit Vector Processor Unit |
| 9 | Design Support |
| 10 | Mechanical Data |

SN74ACT8832A 32-BIT REGISTERED ALU

Dxxx, JANUARY 1990

- 40-ns Cycle Time
- Low-Power 1- μ m EPIC™ CMOS
- Three-Port I/O Architecture
- 64-Word by 36-Bit Register File
- Simultaneous ALU and Register Operations
- Configurable as Quad 8-Bit or Dual 16-Bit Single Instruction, Multiple Data Machine
- Parity Generation/Checking

description

The SN74ACT8832A is a 32-bit registered ALU (Arithmetic/Logic Unit) that can operate at 25 MHz and 25 MIPS (million instructions per second). Most instructions can be performed in a single cycle. The 'ACT8832A was designed for applications that require high-speed logical, arithmetic, and shift operations and bit/byte manipulations.

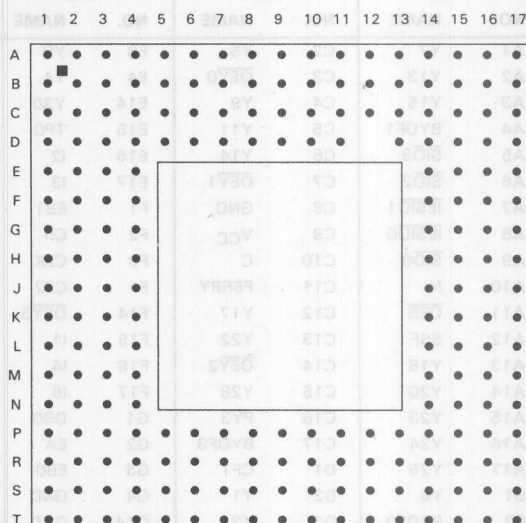
The 'ACT8832A can act as host CPU or can accelerate a host microprocessor. In high-performance graphics systems, the 'ACT8832A generates display-list memory addresses and controls the display buffer. In I/O controller applications, the 'ACT8832A performs high-speed comparisons to initialize and end data transfers.

A three-operand, 64-word by 36-bit register file allows the 'ACT8832A to create an instruction and store the previous result in a single cycle.

The 'ACT8832A registered ALU holds a primary position in the Texas Instruments family of innovative 32-bit LSI devices. Compatible with the SN74AS888 architecture and instruction set, the 'ACT8832A performs as a high-speed microprogrammable 32-bit registered ALU that can also be configured to operate as two 16-bit ALUs or four 8-bit ALUs in single-instruction, multiple-data (SIMD) mode.

Besides introducing the 'ACT8832A, this data sheet discusses basic concepts of microprogrammed architecture and the support tools available for system development. Details of the 'ACT8832A architecture and instruction set are presented. Pin descriptions and assignments for the 'ACT8832A are also presented.

GB . . . PACKAGE (TOP VIEW)



Note: The location between pins A1, A2, B1, and B2 is marked for indexing purposes.

EPIC is a trademark of Texas Instruments Incorporated.

PRODUCTION DATA documents contain information current as of publication date. Products conform to specifications per the terms of Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.

TEXAS
INSTRUMENTS

POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

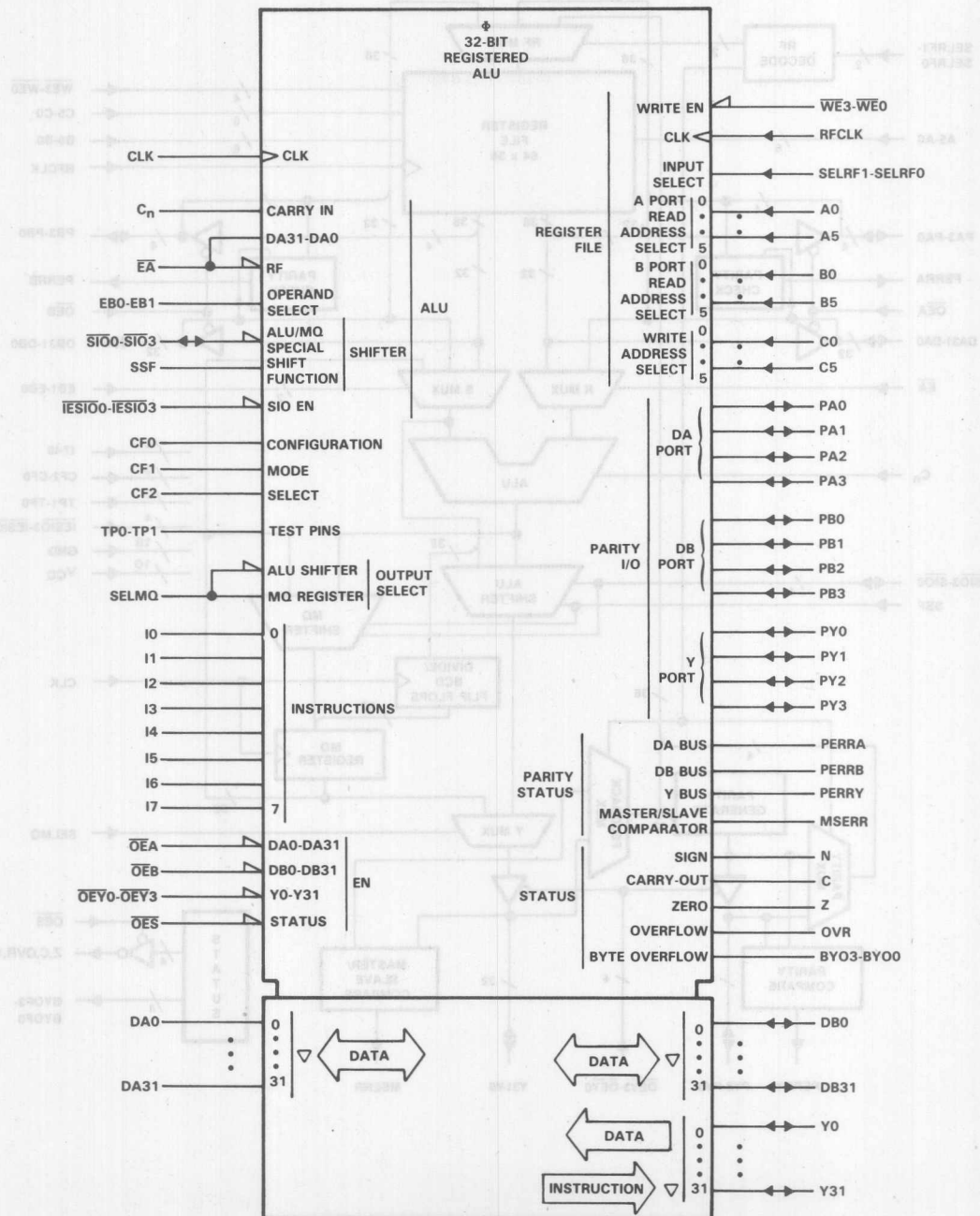
SN74ACT8832A **32-BIT REGISTERED ALU**

PIN ASSIGNMENTS

| PIN | | PIN | | PIN | | PIN | | PIN | | PIN | |
|-----|--------|-----|--------|-----|------|-----|------|-----|-------|-----|--------|
| NO. | NAME | NO. | NAME | NO. | NAME | NO. | NAME | NO. | NAME | NO. | NAME |
| A1 | Y7 | C2 | Y5 | E3 | Y0 | J15 | DA28 | P1 | DA5 | S1 | DB10 |
| A2 | Y13 | C3 | OEY0 | E4 | Y4 | J16 | DA27 | P2 | DB8 | S2 | DB15 |
| A3 | Y15 | C4 | Y9 | E14 | Y30 | J17 | DA29 | P3 | DB12 | S3 | DA10 |
| A4 | BYOF1 | C5 | Y11 | E15 | TP0 | K1 | DB6 | P4 | DA9 | S4 | DA13 |
| A5 | STO3 | C6 | Y14 | E16 | I2 | K2 | DB7 | P5 | DA15 | S5 | PERRA |
| A6 | STO2 | C7 | OEY1 | E17 | I3 | K3 | DA0 | P6 | A5 | S6 | A3 |
| A7 | IESIO1 | C8 | GND | F1 | EB1 | K4 | GND | P7 | A1 | S7 | WE0 |
| A8 | IESIO0 | C9 | VCC | F2 | Cn | K14 | GND | P8 | VCC | S8 | WE3 |
| A9 | STO0 | C10 | C | F3 | CLK | K15 | DA24 | P9 | GND | S9 | RFCLK |
| A10 | N | C11 | PERRY | F4 | CF2 | K16 | DA25 | P10 | C4 | S10 | B4 |
| A11 | OES | C12 | Y17 | F14 | OEY3 | K17 | DA26 | P11 | PERRB | S11 | B2 |
| A12 | SSF | C13 | Y22 | F15 | I1 | L1 | PB0 | P12 | GND | S12 | C3 |
| A13 | Y18 | C14 | OEY2 | F16 | I4 | L2 | DA2 | P13 | DB22 | S13 | C0 |
| A14 | Y20 | C15 | Y28 | F17 | I6 | L3 | VCC | P14 | DA16 | S14 | DB17 |
| A15 | Y23 | C16 | PY3 | G1 | DB0 | L4 | GND | P15 | DA18 | S15 | DB20 |
| A16 | Y24 | C17 | BYOF3 | G2 | EA | L14 | GND | P16 | DA22 | S16 | DB23 |
| A17 | Y25 | D1 | CF1 | G3 | EB0 | L15 | VCC | P17 | DB27 | S17 | DA21 |
| B1 | Y6 | D2 | Y1 | G4 | GND | L16 | DB30 | R1 | PA0 | T1 | DB14 |
| B2 | BYOFO | D3 | Y3 | G14 | GND | L17 | PB3 | R2 | DB11 | T2 | DA8 |
| B3 | Y10 | D4 | PY0 | G15 | I5 | M1 | DA1 | R3 | PB1 | T3 | DA12 |
| B4 | Y12 | D5 | Y8 | G16 | I7 | M2 | DA4 | R4 | DA11 | T4 | DA14 |
| B5 | PY1 | D6 | GND | G17 | PA3 | M3 | DA7 | R5 | PA1 | T5 | OEA |
| B6 | IESIO3 | D7 | GND | H1 | DB2 | M4 | GND | R6 | A4 | T6 | A2 |
| B7 | IESIO2 | D8 | GND | H2 | DB1 | M14 | PA2 | R7 | A0 | T7 | WE1 |
| B8 | STO1 | D9 | VCC | H3 | VCC | M15 | DB26 | R8 | WE2 | T8 | SELRF1 |
| B9 | Z | D10 | GND | H4 | GND | M16 | DB28 | R9 | VCC | T9 | SELRF0 |
| B10 | OVR | D11 | GND | H14 | GND | M17 | DB31 | R10 | B1 | T10 | B5 |
| B11 | MSERR | D12 | No Pin | H15 | VCC | N1 | DA3 | R11 | C2 | T11 | B3 |
| B12 | Y16 | D13 | BYOF2 | H16 | DA31 | N2 | DA6 | R12 | OEB | T12 | B0 |
| B13 | Y19 | D14 | Y27 | H17 | DA30 | N3 | DB9 | R13 | DB18 | T13 | C5 |
| B14 | Y21 | D15 | Y31 | J1 | DB3 | N4 | DB13 | R14 | DB21 | T14 | C1 |
| B15 | PY2 | D16 | TP1 | J2 | DB4 | N14 | DA19 | R15 | PB2 | T15 | DB16 |
| B16 | Y26 | D17 | IO | J3 | DB5 | N15 | DA23 | R16 | DA20 | T16 | DB19 |
| B17 | Y29 | E1 | SELMQ | J4 | VCC | N16 | DB25 | R17 | DB24 | T17 | DA17 |
| C1 | Y2 | E2 | CF0 | J14 | VCC | N17 | DB29 | | | | |

SN74ACT8832A 32-BIT REGISTERED ALU

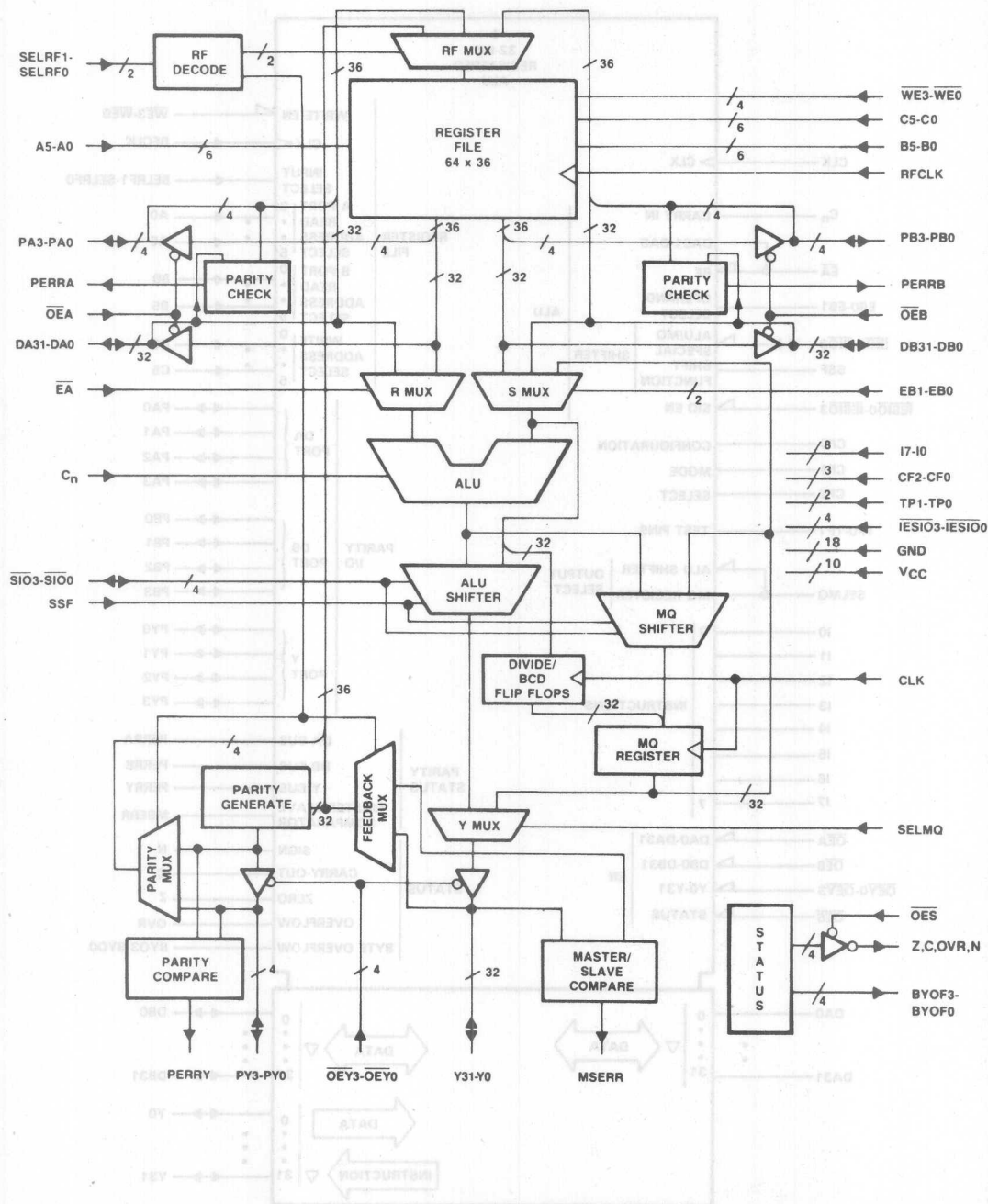
logic symbol†



†This symbol is in accordance with ANSI/IEEE Std. 91-1984.

SN74ACT8832A **32-BIT REGISTERED ALU**

functional block diagram



SN74ACT8832A
32-BIT REGISTERED ALU

TERMINAL FUNCTIONS

| PIN | | I/O | DESCRIPTION | IO | NAME |
|-------|-----|-----|---|----|------|
| NAME | NO. | | | | |
| A0 | R7 | I | Register file A port read address select | | DA30 |
| A1 | P7 | | | | DA31 |
| A2 | T6 | | | | DA32 |
| A3 | S6 | | | | DA33 |
| A4 | R6 | | | | DA34 |
| A5 | P6 | | | | DA35 |
| B0 | T12 | I | Register file B port read address select | | DA36 |
| B1 | R10 | | | | DA37 |
| B2 | S11 | | | | DA38 |
| B3 | T11 | | | | DA39 |
| B4 | S10 | | | | DA40 |
| B5 | T10 | | | | DA41 |
| BYOF0 | B2 | O | Status signals indicate overflow conditions in certain data bytes | | DB0 |
| BYOF1 | A4 | | | | DB1 |
| BYOF2 | D13 | | | | DB2 |
| BYOF3 | C17 | | | | DB3 |
| C | C10 | O | Status signal representing carry out condition | | DB4 |
| C0 | S13 | I | Register file write address select | | DB5 |
| C1 | T14 | | | | DB6 |
| C2 | R11 | | | | DB7 |
| C3 | S12 | | | | DB8 |
| C4 | P10 | | | | DB9 |
| C5 | T13 | | | | DB10 |
| CF0 | E2 | I | Configuration mode select, single 32-bit, two 16-bit, or four 8-bit ALUs | | DB11 |
| CF1 | D1 | | | | DB12 |
| CF2 | F4 | | | | DB13 |
| Cn | F2 | I | ALU carry input | | DB14 |
| CLK | F3 | I | Clocks synchronous registers on positive edge | | DB15 |
| DA0 | K3 | I/O | A port data bus. Outputs register data ($\overline{OE}A = 0$) or inputs external data ($\overline{OE}A = 1$). | | DB16 |
| DA1 | M1 | | | | DB17 |
| DA2 | L2 | | | | DB18 |
| DA3 | N1 | | | | DB19 |
| DA4 | M2 | | | | DB20 |
| DA5 | P1 | | | | DB21 |
| DA6 | N2 | | | | DB22 |
| DA7 | M3 | | | | DB23 |
| DA8 | T2 | | | | DB24 |
| DA9 | P4 | | | | DB25 |
| DA10 | S3 | | | | DB26 |
| DA11 | R4 | | | | DB27 |
| DA12 | T3 | | | | DB28 |
| DA13 | S4 | | | | DB29 |
| DA14 | T4 | | | | DB30 |
| DA15 | P5 | | | | DB31 |
| DA16 | P14 | | | I | EA |
| DA17 | T17 | | | I | EB |
| DA18 | P15 | | | | EB1 |
| DA19 | N14 | | | | EB2 |



SN74ACT8832A **32-BIT REGISTERED ALU**

TERMINAL FUNCTIONS (continued)

| PIN | | I/O | DESCRIPTION | PIN | NAME |
|------------------|-----|-----|--|-----|------|
| NAME | NO. | | | | |
| DA20 | R16 | I/O | A port data bus. Outputs register data ($\overline{OE}A = 0$) or inputs external data ($\overline{OE}A = 1$). | 80 | 80 |
| DA21 | S17 | | | 81 | 81 |
| DA22 | P16 | | | 82 | 82 |
| DA23 | N15 | | | 83 | 83 |
| DA24 | K15 | | | 84 | 84 |
| DA25 | K16 | | | 85 | 85 |
| DA26 | K17 | | | 86 | 86 |
| DA27 | J16 | | | 87 | 87 |
| DA28 | J15 | | | 88 | 88 |
| DA29 | J17 | | | 89 | 89 |
| DA30 | H17 | | | 90 | 90 |
| DA31 | H16 | | | 91 | 91 |
| DB0 | G1 | | | 92 | 92 |
| DB1 | H2 | | | 93 | 93 |
| DB2 | H1 | | | 94 | 94 |
| DB3 | J1 | | | 95 | 95 |
| DB4 | J2 | | | 96 | 96 |
| DB5 | J3 | | | 97 | 97 |
| DB6 | K1 | | | 98 | 98 |
| DB7 | K2 | | | 99 | 99 |
| DB8 | P2 | | | 100 | 100 |
| DB9 | N3 | | | 101 | 101 |
| DB10 | S1 | | | 102 | 102 |
| DB11 | R2 | | | 103 | 103 |
| DB12 | P3 | I/O | B port data bus. Outputs register data ($\overline{OE}B = 0$) or used to input external data ($\overline{OE}B = 1$). | 104 | 104 |
| DB13 | N4 | | | 105 | 105 |
| DB14 | T1 | | | 106 | 106 |
| DB15 | S2 | | | 107 | 107 |
| DB16 | T15 | | | 108 | 108 |
| DB17 | S14 | | | 109 | 109 |
| DB18 | R13 | | | 110 | 110 |
| DB19 | T16 | | | 111 | 111 |
| DB20 | S15 | | | 112 | 112 |
| DB21 | R14 | | | 113 | 113 |
| DB22 | P13 | | | 114 | 114 |
| DB23 | S16 | | | 115 | 115 |
| DB24 | R17 | | | 116 | 116 |
| DB25 | N16 | | | 117 | 117 |
| DB26 | M15 | | | 118 | 118 |
| DB27 | P17 | | | 119 | 119 |
| DB28 | M16 | | | 120 | 120 |
| DB29 | N17 | | | 121 | 121 |
| DB30 | L16 | | | 122 | 122 |
| DB31 | M17 | | | 123 | 123 |
| E \overline{A} | G2 | I | ALU input operand select. High state selects external DA bus and low state selects register file. | 124 | 124 |
| EBO | G3 | I | ALU input operand select. Selects between register file, external DB port and MQ register | 125 | 125 |
| EB1 | F1 | | | 126 | 126 |



TERMINAL FUNCTIONS (continued)

| PIN NAME | NO. | I/O | DESCRIPTION |
|-------------|-----|-----|--|
| GND | C8 | | |
| GND | D6 | | |
| GND | D7 | | |
| GND | D8 | | |
| GND | D10 | | |
| GND | D11 | | |
| GND | G4 | | |
| GND | G14 | | |
| GND | H4 | | Ground pins. All ground pins must be used. |
| GND | H14 | | |
| GND | K4 | | |
| GND | K14 | | |
| GND | L4 | | |
| GND | L14 | | |
| GND | M4 | | |
| GND | P9 | | |
| GND | P12 | | |
| I0 | D17 | | |
| I1 | F15 | | |
| I2 | E16 | | |
| I3 | E17 | I | Instruction input |
| I4 | F16 | | |
| I5 | G15 | | |
| I6 | F17 | | |
| I7 | G16 | | |
| IESIO0 | A8 | | |
| IESIO1 | A7 | I | Shift pin enables, increases system speed and reduces bus conflict, active low |
| IESIO2 | B7 | | |
| IESIO3 | B6 | | |
| MSERR | B11 | O | Master Slave Error pin, indicates error between data at Y output MUX and external Y port |
| N | A10 | O | Output status signal representing sign condition |
| OE A | T5 | I | DA bus enable, active low |
| OE B | R12 | I | DB bus enable, active low |
| OE S | A11 | I | Status enable, active low |
| OEY0 | C3 | | |
| OEY1 | C7 | I | Y bus output enable, active low |
| OEY2 | C14 | | |
| OEY3 | F14 | | |
| OVR | B10 | O | Output status signal represents overflow condition |
| PA0 | R1 | | |
| PA1 | R5 | I/O | Parity bits port for DA data |
| PA2 | M14 | | |
| PA3 | G17 | | |
| PB0 | L1 | | |
| PB1 | R3 | I/O | Parity bits port for DB data |
| PB2 | R15 | | |
| PB3 | L17 | | |

SN74ACT8832A **32-BIT REGISTERED ALU**

TERMINAL FUNCTIONS (continued)

| PIN | | I/O | DESCRIPTION | NO. | I/O | DESCRIPTION | NO. | I/O |
|--------|-----|-----|---|-----|-----|-------------|-----|-----|
| NAME | NO. | | | | | | | |
| PERRA | S5 | O | DA data parity error, signals error if an even parity check fails for any byte | | | | | |
| PERRB | P11 | O | DB data parity error, signals error if an even parity check fails for any byte | | | | | |
| PERRY | C11 | O | Y data parity error, signals error if an even parity check fails for any byte | | | | | |
| PY0 | D4 | I/O | Y port parity data, input and output | | | | | |
| PY1 | B5 | | | | | | | |
| PY2 | B15 | | | | | | | |
| PY3 | C16 | | | | | | | |
| RFCLK | S9 | I | Register File Clock, allows multiple writes to be performed in one master clock cycle | | | | | |
| SELMQ | E1 | I | MQ register select, selects output of ALU shifter or MQ register to be placed on Y bus | | | | | |
| SELRFO | T9 | I | Register File select. Controls selection of the Register File(RF) inputs by the RF MUX | | | | | |
| SELRF1 | T8 | | | | | | | |
| SIO0 | A9 | I/O | Bidirectional shift pin, active low | | | | | |
| SIO1 | B8 | | | | | | | |
| SIO2 | A6 | | | | | | | |
| SIO3 | A5 | | | | | | | |
| SSF | A12 | I | Special Shift Function, implements conditional shift algorithms | | | | | |
| TP0 | E15 | | Test pins, supports system testing. Tied high for normal operation. | | | | | |
| TP1 | D16 | | | | | | | |
| VCC | C9 | | Supply voltage (5 V) | | | | | |
| VCC | D9 | | | | | | | |
| VCC | H3 | | | | | | | |
| VCC | H15 | | | | | | | |
| VCC | J4 | | | | | | | |
| VCC | J14 | | | | | | | |
| VCC | L3 | | | | | | | |
| VCC | L15 | | | | | | | |
| VCC | P8 | | | | | | | |
| VCC | R9 | | | | | | | |
| WE0 | S7 | | Write Enable. Data is written into RF when write enables are low and a low to high Register File Clock (RFCLK) transition occurs. Active low. | | | | | |
| WE1 | T7 | | | | | | | |
| WE2 | R8 | | | | | | | |
| WE3 | S8 | | | | | | | |
| Y0 | E3 | I/O | Y port data bus | | | | | |
| Y1 | D2 | | | | | | | |
| Y2 | C1 | | | | | | | |
| Y3 | D3 | | | | | | | |
| Y4 | E4 | | | | | | | |
| Y5 | C2 | | | | | | | |
| Y6 | B1 | | | | | | | |
| Y7 | A1 | | | | | | | |
| Y8 | D5 | | | | | | | |
| Y9 | C4 | | | | | | | |
| Y10 | B3 | | | | | | | |
| Y11 | C5 | | | | | | | |
| Y12 | B4 | | | | | | | |
| Y13 | A2 | | | | | | | |
| Y14 | C6 | | | | | | | |
| Y15 | A3 | | | | | | | |

TERMINAL FUNCTIONS (continued)

| PIN NAME | NO. | I/O | DESCRIPTION |
|-------------|-----|-----|--|
| Y16 | B12 | | |
| Y17 | C12 | | |
| Y18 | A13 | | |
| Y19 | B13 | | |
| Y20 | A14 | | |
| Y21 | B14 | | |
| Y22 | C13 | | |
| Y23 | A15 | | |
| Y24 | A16 | I/O | Y port data bus |
| Y25 | A17 | | |
| Y26 | B16 | | |
| Y27 | D14 | | |
| Y28 | C15 | | |
| Y29 | B17 | | |
| Y30 | E14 | | |
| Y31 | D15 | | |
| Z | B9 | O | Output status signal represents zero condition |

understanding microprogrammed architecture

Figure 1 shows a simple microprogrammed system. The three basic components are an arithmetic/logic unit, a microsequencer, and a memory. The program that resides in this memory is commonly called the microprogram, while the memory itself is referred to as a micromemory or control store. The ALU performs all the required operations on data brought in from the external environment (main memory or peripherals, for example). The sequencer is dedicated to generating the next micromemory address from which a microinstruction is to be fetched. The sequencer and the ALU operate in parallel so that data processing and next-address generation are carried out concurrently.

The microprogram instruction, or microinstruction, consists of control information to the ALU and the sequencer. The microinstruction consists of a number of fields of code that directly access and control the ALU, registers, bus transceivers, multiplexers, and other system components. This high degree of programmability in a parallel architecture offers greater speed and flexibility than a typical microprocessor, although the microinstruction serves the same purpose as a microprocessor opcode: it specifies control information by which the user is able to implement desired data processing operations in a specified sequence. The microinstruction cycle is synchronized to a system clock by latching the instruction in the microinstruction, or pipeline, register once for each clock cycle. Status results are collected in a status register which the sequencer samples to produce conditional branches within the microprogram.

SN74ACT8832A 32-BIT REGISTERED ALU

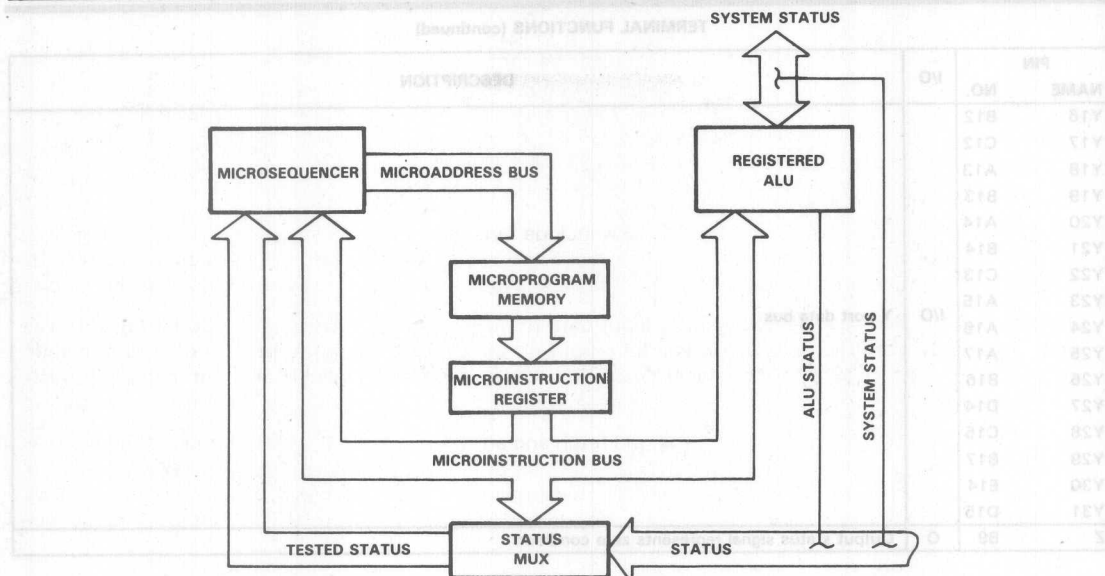


FIGURE 1. MICROPROGRAMMED SYSTEM BLOCK DIAGRAM

design support

TI's 'ACT8832A 32-bit registered ALU is supported by a variety of tools developed to aid in design evaluation and verification. These tools will streamline all stages of the design process, from assessing the operation and performance of the 'ACT8832A to evaluating a total system application. The tools include a functional model, behavioral model, and microcode development software and hardware. Section 9 of this manual provides specific information on the design tools supporting TI's '8800 Family.

systems expertise

Texas Instruments Datapath VLSI Products Systems Engineering group is available to help designers analyze TI's high-performance VLSI products, such as the 'ACT8832A 32-bit registered ALU. The group works directly with designers to provide ready answers to device-related questions and also prepares a variety of applications documentation.

The group may be reached in Dallas, at (214) 997-3970.

registered ALU

The 'ACT8832A is a 32-bit registered ALU that can be configured to operate as four 8-bit ALUs, two 16-bit ALUs, or a single 32-bit ALU. The processor instruction set is 100 percent upwardly compatible with the 'AS888 and includes 13 arithmetic and logical functions with 8 conditional shifts, multiplication, division, normalization, add and subtract immediate, bit and byte operations, and data conversions such as BCD, excess-3, and sign magnitude. New instructions permit internal flip-flops controlling BCD and divide operations to be loaded or read.

Additional functions added to the 'ACT8832A include byte parity and master/slave operation. Parity is checked at the three data input ports and generated at the Y output port. The 64-word register file is 36 bits wide to permit storage of the parity bits. Master/slave comparator circuitry is provided at the Y port.

The DA and DB ports can simultaneously input data to the ALU and the 64-word by 36-bit register file. Data and parity from the register file can be output on the DA and DB ports. Results of ALU and shift operations are output at the bidirectional Y port. The Y port can also be used in an input mode to furnish external data to the register file or during master/slave operation as an input to the master/slave comparator.

Three 6-bit address ports allow a two-operand fetch and an operand write to be performed at the register file simultaneously. An MQ shifter and MQ register can also be configured to function independently to implement double-precision 8-bit, 16-bit, and 32-bit shift operations. An internal ALU bypass path increases the speeds of multiply, divide and normalize instructions. The path is also used by 'ACT8832A instructions that permit bits and bytes to be manipulated.

architecture

Control input signals are summarized in Table 1. Data flow and details of the functional elements are presented in the following paragraphs (see the functional block diagram).

TABLE 1. 'ACT8832A RESPONSE TO CONTROL INPUTS

| SIGNAL | HIGH | LOW |
|---------------|------------------------------|--|
| CF2-CF0 | See Table 5 | See Table 5 |
| EA | Selects external DA bus | Selects register file |
| EB1-EB0 | See Table 3 | See Table 3 |
| IESIO3-IESIO0 | Normal operation | Force corresponding SIO inputs to high impedance |
| I7-I0 | See Table 9 | See Table 9 |
| MQSEL | Selects MQ register | Selects ALU |
| OE \bar{A} | Inhibits DA and PA output | Enables DA and PA output |
| OE \bar{B} | Inhibits DB and PB output | Enables DB and PB output |
| OEY3-OEY0 | Inhibits Y and PY outputs | Enables Y and PY outputs |
| SELRF1-SELRF0 | See Table 2 | See Table 2 |
| SSF | Selects shifted ALU output | Selects ALU (unshifted) output |
| TP1-TP0 | See Table 8 | See Table 8 |
| WE3-WE0 | Inhibits register file write | Byte enables for register file write (0 = LSB) |

SN74ACT8832A **32-BIT REGISTERED ALU**

data flow

As shown in Figure 2, data enters the 'ACT8832A from three primary sources: the bidirectional Y port, which is used in an input mode to pass data to the register file; and the bidirectional DA and DB ports, used to input data to the register file or the R and S buses serving the ALU. Three associated I/O ports (PY, PA, and PB) are provided for associated parity data input and output.

Data is input to the ALU through two multiplexers: R MUX, which selects the R bus operand from the DA port or the register file addressed by A5-A0; and S MUX, which selects data from the DB port, the register file addressed by B5-B0, or the multiplier-quotient (MQ) register.

The result of the ALU operation is passed to the ALU shifter, where it is shifted or passed without shift to the Y bus for possible output from the 'ACT8832A and to the feedback MUX for possible storage in the internal register file. The MQ shifter, which operates in parallel with the ALU shifter, can be loaded from the ALU or the MQ register. The MQ shift result is passed to the MQ register, where it can be routed through the S MUX to the ALU or to the Y MUX for output from the chip.

An internal bypass path allows data from the S MUX to be loaded directly into the ALU shifter or the divide/BCD flip-flops. Data from the divide/BCD flip-flops can be output via the MQ register.

Data can be output from the three bidirectional ports, Y, DA, and DB, and their associated parity ports, PY, PA, and PB. DA and DB can also be used to read ALU input data on the R and S buses for debug or other special purposes.

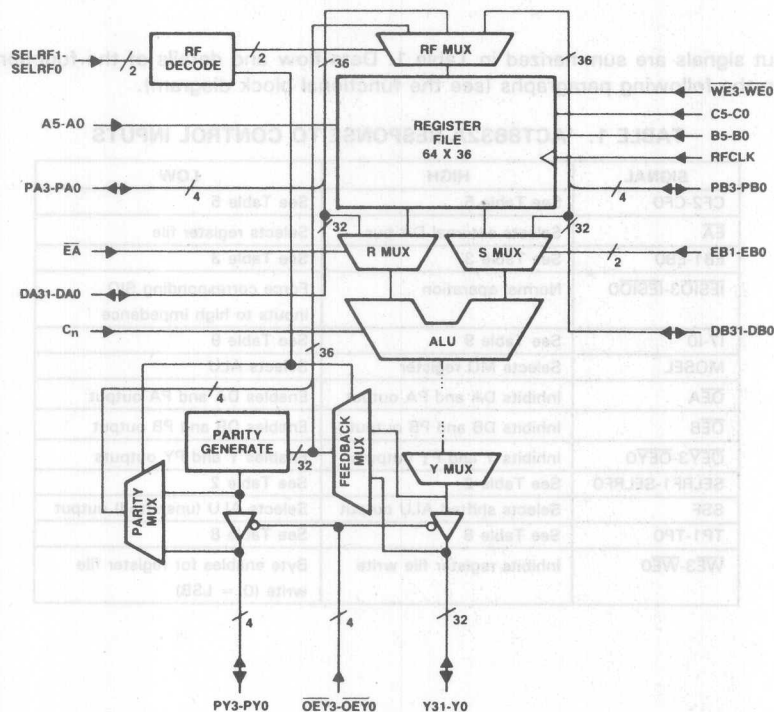


FIGURE 2. DATA I/O

architectural elements

three-port register file

The register file is 36 bits wide, permitting storage of a 32-bit data word with its associated parity bits. The 64 registers are accessed by three address ports. C5-C0 address the destination register during write operations; A5-A0 and B5-B0 address any two registers during read operations. The address buses are also used to furnish immediate data to the ALU: A3-A0 to provide constant data for the add and subtract immediate instructions; C3-C0 and A3-A0 to provide masks for set, reset, and test bit operations.

Data is written into the register file when the write enable is low and a low-to-high register file clock (RFCLK) transition occurs. The separate register file clock allows multiple writes to be performed in one master clock cycle, allowing processors in multi-processor environments to update one another's internal register files during a single cycle.

Four write enable inputs are provided to allow separate control of data inputs in a byte-oriented system. WE3 is the write enable for the most significant byte.

Register file inputs are selected by the RF MUX under the control of two register file select signals, SELRF1 and SELRF0, shown in Table 2 (see also Table 4).

TABLE 2. MUX SELECT INPUTS

| SELRF1 | SELRF0 | SOURCE |
|--------|--------|-------------------|
| 0 | 0 | External DA input |
| 0 | 1 | External DB input |
| 1 | 0 | Y-output MUX |
| 1 | 1 | External Y port |

R and S multiplexers

ALU inputs are selected by the R and S multiplexers. Controls which affect operand selection for instructions other than those using constants or masks are shown in Table 3.

TABLE 3. ALU SOURCE OPERAND SELECTS

| R-BUS OPERAND SELECT EA | S-BUS OPERAND SELECT EB1-EB0 | RESULT DESTINATION | ←SOURCE OPERAND |
|----------------------------------|---------------------------------------|-----------------------|-----------------------------------|
| 0 | | R bus | ←Register file addressed by A5-A0 |
| 1 | | R bus | ←DA port |
| | 0 0 | S bus | ←Register file addressed by B5-B0 |
| | 1 0 | S bus | ←DB port |
| | X 1 | S bus | ←MQ register |

data input and output ports

The DA and DB ports can be used to load the S and/or R multiplexers from an external source or to read S or R bus outputs from the register file. The Y port can be used to load the register file and to output the next address selected by the Y output multiplexer. Tables 9 and 10 describe the MUX and output controls which affect DA, DB, and Y.

SN74ACT8832A **32-BIT REGISTERED ALU**

TABLE 4. DESTINATION OPERAND SELECT/ENABLES

| REGISTER FILE WRITE ENABLE WE | Y BUS OUTPUT ENABLE OEY | Y MUX SELECT MQSEL | REGISTER FILE SELECT RFSEL1-RFSELO | DA PORT OUTPUT ENABLE OE A | DB PORT OUTPUT ENABLE OE B | RESULT DESTINATION ← SOURCE |
|---|----------------------------------|--------------------------|---|--|--|--|
| 1 | 0 | 0 | X | X | | Y/PY ← ALU shifter/parity generate |
| 1 | 0 | 1 | X | X | | Y/PY ← MQ register/parity generate |
| 0 | 0 | 0 | 1 | 0 | | Y/PY, RF ← ALU shifter/parity generate |
| 0 | 0 | 1 | 1 | 0 | | Y/PY, RF ← MQ register/parity generate |
| 0 | 1 | X | 1 | 1 | | RF ← External Y/PY |
| 0 | X | X | 0 | 0 | 1 | RF ← External DA/PA |
| 0 | X | X | 0 | 1 | X | RF ← External DB/PB |
| | | | | 0 | | DA/PA ← R bus register file output |
| | | | | 1 | | DA/PA ← Hi-Z |
| | | | | | 0 | DB/PB ← S bus register file output |
| | | | | | 1 | DB/PB ← Hi-Z |

ALU

The ALU can perform seven arithmetic and six logical instructions on the two 32-bit operands selected by the R and S multiplexers. It also supports multiplication, division, normalization, bit and byte operations and data conversion, including excess-3 BCD arithmetic. The 'ACT8832A instruction set is summarized in Table 9.

The 'ACT8832A can be configured to operate as a single 32-bit ALU, two 16-bit ALUs, or four 8-bit ALUs (see Figures 3 and 4). It can also be configured to operate on a 32-bit word formed by adding leading zeros to the 12 least significant bits of R bus data. This is useful in certain IBM™ relative addressing schemes.

Configuration modes are controlled by three CF inputs as shown in Table 5. These signals also select the data from which status signals other than byte overflow will be generated.

TABLE 5. CONFIGURATION MODE SELECTS

| CONTROL INPUTS | | | MODE SELECTED | DATA FROM WHICH STATUS OTHER THAN BYOF WILL BE GENERATED |
|----------------|-----|-----|---------------|---|
| CF2 | CF1 | CF0 | | |
| 0 | 0 | 0 | Four 8-bit | Byte 0 |
| 0 | 0 | 1 | Four 8-bit | Byte 1 |
| 0 | 1 | 0 | Four 8-bit | Byte 2 |
| 0 | 1 | 1 | Four 8-bit | Byte 3 |
| 1 | 0 | 0 | Two 16-bit | Least significant 16-bit word |
| 1 | 0 | 1 | Two 16-bit | Most significant 16-bit word |
| 1 | 1 | 0 | One 32-bit | 32-bit word |
| 1 | 1 | 1 | Masked 32-bit | 32-bit word |

IBM is a trademark of International Business Machines Corporation.

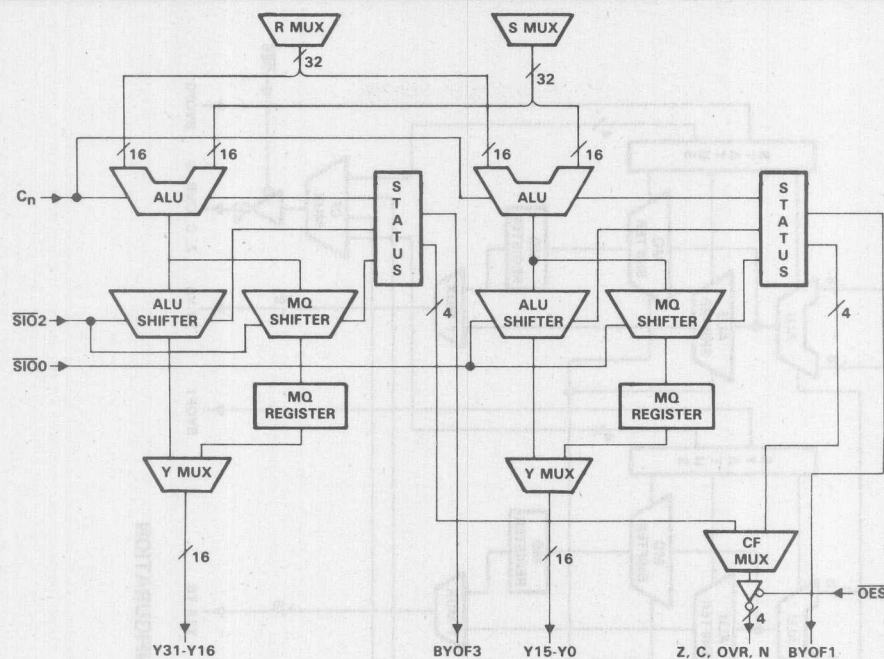


FIGURE 3. 16-BIT CONFIGURATION

ALU and MQ shifters

The ALU and MQ shifters are used in all of the shift, multiply, divide and normalize functions. They can be used independently for single-precision or concurrently for double-precision shifts. Shifts can be made conditional, using the Special Shift Function (SSF) pin.

bidirectional serial I/O pins

Four bidirectional $\overline{\text{SIO}}$ pins are provided to supply an end fill bit for certain shift instructions. These pins may also be used to read bits that are shifted out of the ALU or MQ shifters during certain instructions. Use of the $\overline{\text{SIO}}$ pins as inputs or outputs is summarized in Table 11.

The four pins allow separate control of end fill inputs in configurations other than 32-bit mode (see Table 6 and the functional block diagram).

TABLE 6. DATA DETERMINING $\overline{\text{SIO}}$ INPUT

| SIGNAL | CORRESPONDING WORD, PARTIAL WORD OR BYTE | | |
|--------------------------|--|------------------------|------------|
| | 32-BIT MODE | 16-BIT MODE | 8-BIT MODE |
| $\overline{\text{SIO}}3$ | — | — | Byte 3 |
| $\overline{\text{SIO}}2$ | — | most significant word | Byte 2 |
| $\overline{\text{SIO}}1$ | — | — | Byte 1 |
| $\overline{\text{SIO}}0$ | 32-bit word | least significant word | Byte 0 |

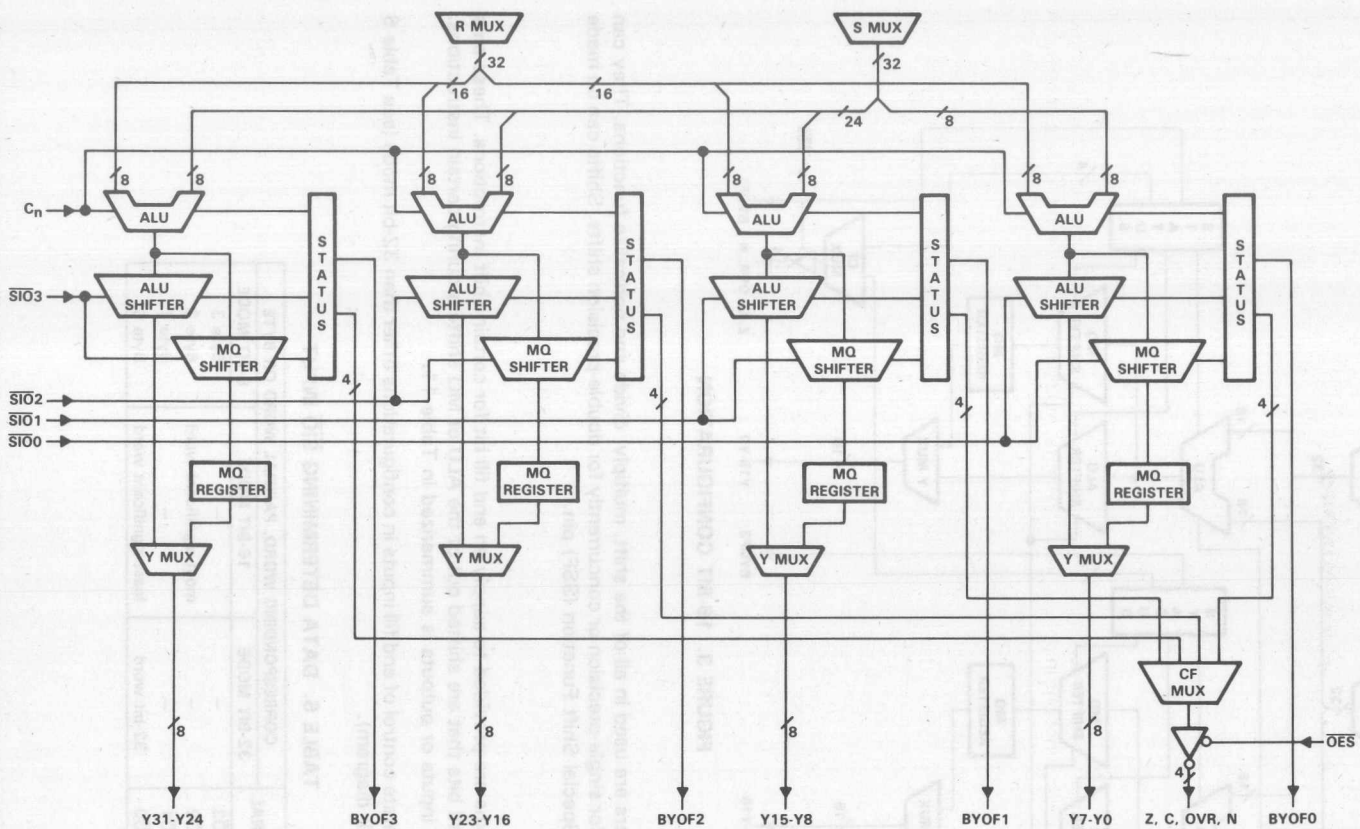


FIGURE 4. 8-BIT CONFIGURATION

To increase system speed and reduce bus conflict, four $\overline{\text{SIO}}$ input enables ($\overline{\text{IESIO3}}$ - $\overline{\text{IESIO0}}$) are provided. A low on these enables will override internal pull-up resistor logic and force the corresponding $\overline{\text{SIO}}$ pins to the high-impedance state required before an input signal can appear on the signal line. If the $\overline{\text{SIO}}$ enables are not used, this condition is generated internally in the chip. Use of the enables allow internal decoding to be bypassed, resulting in faster speeds.

The $\overline{\text{IESIO}}$ s are defaulted to a high because of internal pull-up resistors. When an $\overline{\text{SIO}}$ pin is used as an output, a low on its corresponding $\overline{\text{IESIO}}$ pin would force $\overline{\text{SIO}}$ to a high impedance state. The output would then be lost, but the internal operation of the chip would not be affected.

MQ register

Data from the MQ shifter is written into the MQ register when a low-to-high transition occurs on clock CLK. The register has specific functions in double precision shifts, multiplication, division and data conversion algorithms and can also be used as a temporary storage register. Data from the register file and the DA and DB buses can be passed to the MQ register through the ALU.

The Y bus contains the output of the ALU shifter if SELMQ is low and the output of the MQ register if SELMQ is high. If $\overline{\text{OEY}}$ is low, ALU or MQ shifter output will be passed to the Y port; if $\overline{\text{OEY}}$ is high, the Y port becomes an input to the feedback MUX.

conditional shift pin

Conditional shifting algorithms may be implemented using the SSF pin under hardware or firmware control. If the SSF pin is high or floating, the shifted ALU output will be sent to the output buffers. If the SSF pin is pulled low externally, the ALU result will be passed directly to the output buffers, and MQ shifts will be inhibited. Conditional shifting is useful for scaling inputs in data arrays or in signal processing algorithms.

master/slave comparator

A master/slave comparator is provided to compare data bytes from the Y output MUX with data bytes on the external Y port when $\overline{\text{OEY}}$ is high. If the data are not equal, a high signal is generated on the master slave error output pin (MSERR). A similar comparator is provided for the Y parity bits.

divide/BCD flip-flops

Internal multiply/divide flip-flops are used by certain multiply and divide instructions to maintain status between instructions. Internal excess-3 BCD flip-flops preserve the carry from each nibble in excess-3 BCD operations. The BCD flip-flops are affected by all instructions except NOP and are cleared when a CLR instruction is executed. The flip-flops can be loaded and read externally using instructions LOADFF and DUMPFF (see Table 9). This feature permits an iterative arithmetic operation such as multiplication or division to be interrupted immediately so that an external interrupt can be processed.

status

Eight status output signals are generated by the 'ACT8832A. Four signals (BYOF3-BYOF0) indicate overflow conditions in certain data bytes (see Table 7). The others represent sign (N), zero (ZERO), carry-out (Cout) and overflow (OVR). N, ZERO, Cout, and OVR are generated from data selected by the mode configuration controls (CF2-CF0) as shown in Table 5.

Carry-out is evaluated after each ALU operation. Sign and zero status are evaluated after ALU shift operation. Overflow (OVR) is determined by ORing the overflow result from the ALU with the overflow result from the ALU shifter.

SN74ACT8832A **32-BIT REGISTERED ALU**

TABLE 7. DATA DETERMINING BYOF OUTPUTS

| SIGNAL | CORRESPONDING WORD, PARTIAL WORD OR BYTE | | |
|--------|--|------------------------|------------|
| | 32-BIT MODE | 16-BIT MODE | 8-BIT MODE |
| BYOF3 | 32-bit word | most significant word | Byte 3 |
| BYOF2 | — | — | Byte 2 |
| BYOF1 | — | least significant word | Byte 1 |
| BYOF0 | — | — | Byte 0 |

input data parity check

An even parity check is performed on each byte of input data at the DA, DB and Y ports. The check is performed by counting the number of ones in each byte and its corresponding parity bit. Parity bits are input on PA for DA data, PB for DB data and PYF or Y data. PA0, PB0 and PY0 are the parity bits for the least significant bytes of DA, DB and Y, respectively. If the result of the parity count is odd for any byte, a high appears at the parity error output pin (PERRA for DA data, PERRB for DB data, PERRY for Y data).

test pins

Two pins, TP1-TP0, support system testing. These may be used, for example, to place all outputs in a high-impedance state, isolating the chip from the rest of the system (see Table 8).

TABLE 8. TEST PIN INPUTS

| TP1 | TP0 | RESULT |
|-----|-----|---|
| 0 | 0 | All outputs and I/Os forced low |
| 1 | 0 | All outputs and I/Os forced high |
| 0 | 1 | All outputs and I/Os placed in a high impedance state |
| 1 | 1 | Normal operation (default state) |

instruction set overview

Bits I7-I0 are used as instruction inputs to the 'ACT8832A. Table 9 lists all instructions divided into five groups with their opcodes and mnemonics.

Table 9. 'ACT8832A INSTRUCTION SET

| GROUP 1 INSTRUCTIONS | | |
|------------------------------------|----------|-------------------------------------|
| INSTRUCTION BITS 13-10 (HEX) | MNEMONIC | FUNCTION |
| 0 | | Used to access Group 4 instructions |
| 1 | ADD | $R + S + Cn$ |
| 2 | SUBR | $\bar{R} + S + Cn$ |
| 3 | SUBS | $R + \bar{S} + Cn$ |
| 4 | INCS | $S + Cn$ |
| 5 | INCNS | $\bar{S} + Cn$ |
| 6 | INCR | $R + Cn$ |
| 7 | INCNR | $\bar{R} + Cn$ |
| 8 | | Used to access Group 3 instructions |
| 9 | XOR | $R \text{ XOR } S$ |
| A | AND | $R \text{ AND } S$ |
| B | OR | $R \text{ OR } S$ |
| C | NAND | $R \text{ NAND } S$ |
| D | NOR | $R \text{ NOR } S$ |
| E | ANDNR | $\bar{R} \text{ AND } S$ |
| F | | Used to access Group 5 instructions |

| GROUP 2 INSTRUCTIONS | | |
|------------------------------------|----------|---|
| INSTRUCTION BITS 17-14 (HEX) | MNEMONIC | FUNCTION |
| 0 | SRA | Arithmetic right single-precision shift |
| 1 | SRAD | Arithmetic right double-precision shift |
| 2 | SRL | Logical right single-precision shift |
| 3 | SRLD | Logical right double-precision shift |
| 4 | SLA | Arithmetic left single-precision shift |
| 5 | SLAD | Arithmetic left double-precision shift |
| 6 | SLC | Circular left single-precision shift |
| 7 | SLCD | Circular left double-precision shift |
| 8 | SRC | Circular right single-precision shift |
| 9 | SRCD | Circular right double-precision shift |
| A | MQSRA | Arithmetic right shift MQ register |
| B | MQSRL | Logical right shift MQ register |
| C | MQSLL | Logical left shift MQ register |
| D | MQSLC | Circular left shift MQ register |
| E | LOADMQ | Load MQ register |
| F | PASS | Pass ALU to Y |

TABLE 9. ACT8832A INSTRUCTION SET (Continued)

| GROUP 3 INSTRUCTIONS | | |
|------------------------------------|----------|---------------------------------|
| INSTRUCTION BITS 17-10 (HEX) | MNEMONIC | FUNCTION |
| 08 | SET1 | Set bit 1 |
| 18 | SET0 | Set bit 0 |
| 28 | TB1 | Test bit (one) |
| 38 | TB | Test bit (zero) |
| 48 | ABS | Absolute value |
| 58 | SMTC | Sign magnitude/two's complement |
| 68 | ADDI | Add immediate |
| 78 | SUBI | Subtract immediate |
| 88 | BADD | Byte add R to S |
| 98 | BSUBS | Byte subtract S from R |
| A8 | BSUBR | Byte subtract R from S |
| B8 | BINCS | Byte increment S |
| C8 | BINCNS | Byte increment negative S |
| D8 | BXOR | Byte XOR R and S |
| E8 | BAND | Byte AND R and S |
| F8 | BOR | Byte OR R and S |

| GROUP 4 INSTRUCTIONS | | |
|------------------------------------|----------|--|
| INSTRUCTION BITS 17-10 (HEX) | MNEMONIC | FUNCTION |
| 00 | CRC | Cyclic redundancy character accumulation |
| 10 | SEL | Select S or R |
| 20 | SNORM | Single-length normalize |
| 30 | DNORM | Double-length normalize |
| 40 | DIVRF | Divide remainder fix |
| 50 | SDIVQF | Signed divide quotient fix |
| 60 | SMULI | Signed multiply iterate |
| 70 | SMULT | Signed multiply terminate |
| 80 | SDIVIN | Signed divide initialize |
| 90 | SDIVIS | Signed divide start |
| A0 | SDIVI | Signed divide iterate |
| B0 | UDIVIS | Unsigned divide start |
| C0 | UDIVI | Unsigned divide iterate |
| D0 | UMULI | Unsigned multiply iterate |
| E0 | SDIVIT | Signed divide terminate |
| F0 | UDIVIT | Unsigned divide terminate |

TABLE 9. 'ACT8832A INSTRUCTION SET (Continued)

| GROUP 5 INSTRUCTIONS | | |
|------------------------------------|----------|------------------------------|
| INSTRUCTION BITS I7-I0 (HEX) | MNEMONIC | FUNCTION |
| 0F | LOADFF | Load divide/BCD flip-flops |
| 1F | CLR | Clear |
| 2F | CLR | Clear |
| 3F | CLR | Clear |
| 4F | CLR | Clear |
| 5F | DUMPFF | Output divide/BCD flip-flops |
| 6F | CLR | Clear |
| 7F | BCDBIN | BCD to binary |
| 8F | EX3BC | Excess-3 byte correction |
| 9F | EX3C | Excess-3 word correction |
| AF | SDIVO | Signed divide overflow test |
| BF | CLR | Clear |
| CF | CLR | Clear |
| DF | BINEX3 | Binary to excess-3 |
| EF | CLR | Clear |
| FF | NOP | No operation |

Group 1, a set of ALU arithmetic and logic operations, can be combined with the user-selected shift operations in Group 2 in one instruction cycle. The other groups contain instructions for bit and byte operations, division and multiplication, data conversion, and other functions such as sorting, normalization and polynomial code accumulation.

arithmetic/logic instructions with shifts

The seven Group 1 arithmetic instructions operate on data from the R and/or S multiplexers and the carry-in. Carry-out is evaluated after ALU operation; other status pins are evaluated after the accompanying shift operation, when applicable. Group 1 logic instructions do not use carry-in; carry-out is forced to zero.

Possible shift instructions are listed in Group 2. Fourteen single and double precision shifts can be specified, or the ALU result can be passed unshifted to the MQ register or to the specified output destination by using the LOADMQ or PASS instructions. Table 10 lists shift definitions.

TABLE 10. SHIFT DEFINITIONS

| SHIFT TYPE | NOTES |
|------------------|---|
| Left | Moves a bit one position towards the most significant bit |
| Right | Moves a bit one position towards the least significant bit |
| Arithmetic right | Retains the sign unless an overflow occurs, in which case, the sign would be inverted |
| Arithmetic left | May lose the sign bit if an overflow occurs. Zero is filled into the least significant bit unless the bit is set externally |
| Circular right | Fills the least significant bit in the most significant bit position |
| Circular left | Fills the most significant bit in the least significant bit position |
| Logical right | Fills a zero in the most significant bit position unless the bit is forced to one by placing a zero on an \overline{SIO} pin |
| Logical left | Fills a zero in the least significant bit position unless the bit is forced to one by placing a zero on an \overline{SIO} pin |

SN74ACT8832A **32-BIT REGISTERED ALU**

When using the shift registers for double-precision operations, the least significant half should be placed in the MQ register and the most significant half in the ALU for passage to the ALU shifter. An example of a double-precision shift using the ALU and MQ shifters is given in Figure 5.

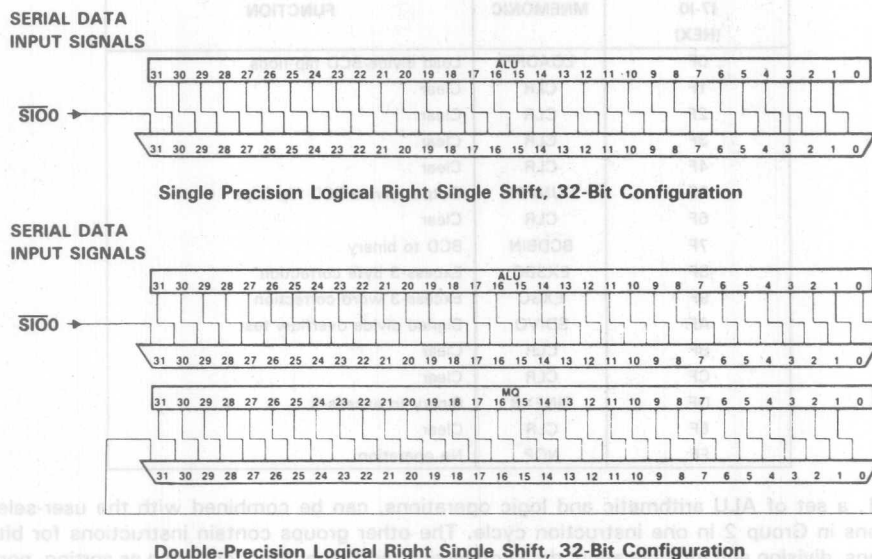


FIGURE 5. SHIFT EXAMPLES, 32-BIT CONFIGURATION

All Group 2 shifts can be made conditional using the conditional shift pin (SSF). If the SSF pin is high or floating, the shifted ALU output will be sent to the output buffers, MQ register, or both. If the SSF pin is pulled low, the ALU result will be passed directly to the output buffers and any MQ shifts will be inhibited.

The bidirectional $\overline{\text{SIO}}$ pins can be used to supply external end fill bits for certain Group 2 shift instructions. When $\overline{\text{SIO}}$ is high or floating, a zero is filled, otherwise a 1 is filled. Table 11 lists instructions that make use of the $\overline{\text{SIO}}$ inputs and identifies input and output functions.

TABLE 10. SHIFT DEFINITIONS

| SHIFT TYPE | NOTES |
|------------------|---|
| Logical left | Is forced to one by placing a zero on an $\overline{\text{SIO}}$ pin |
| Logical right | Fills a zero in the most significant bit position unless the bit is forced to one by placing a zero on an $\overline{\text{SIO}}$ pin |
| Circular left | Fills the most significant bit in the least significant bit position |
| Circular right | Fills the least significant bit in the most significant bit position |
| Arithmetic left | May lose the sign bit if an overflow occurs. Zero is filled into the least significant bit unless the bit is set externally. |
| Arithmetic right | Retains the sign unless an overflow occurs. In which case, the sign would be inverted |
| Right | Moves a bit one position towards the least significant bit |
| Left | Moves a bit one position towards the most significant bit |

TABLE 11. BIDIRECTIONAL SIO PIN FUNCTIONS

| INSTRUCTION BITS 17-10 (HEX) | SIO | | |
|------------------------------------|----------|-----|-----------------------------------|
| | MNEMONIC | I/O | DATA |
| 0* | SRA | O | Shift out |
| 1* | SRAD | O | Shift out |
| 2* | SRL | I | Most significant bit |
| 3* | SRLD | I | Most significant bit |
| 4* | SLA | I | Least significant bit |
| 5* | SLAD | I | Least significant bit |
| 6* | SLC | O | Shifted input to MQ shifter |
| 7* | SLCD | O | Shifted input to MQ shifter |
| 8* | SRC | O | Shifted input to ALU shifter |
| 9* | SRCD | O | Shifted input to ALU shifter |
| A* | MQSRA | O | Shift out |
| B* | MQSRL | I | Most significant bit |
| C* | MQSLL | I | Least significant bit |
| D* | MQSLC | O | Shifted input to MQ shifter |
| 00 | CRC | O | Internally generated end fill bit |
| 20 | SNORM | I | Least significant bit |
| 30 | DNORM | I | Least significant bit |
| 60 | SMULI | O | ALU0 |
| 70 | SMULT | O | ALU0 |
| 80 | SDIVIN | O | Internally generated end fill bit |
| 90 | SDIVIS | O | Internally generated end fill bit |
| A0 | SDIVI | O | Internally generated end fill bit |
| B0 | UDIVIS | O | Internally generated end fill bit |
| C0 | UDIVI | O | Internally generated end fill bit |
| D0 | UMULI | O | Internal input |
| E0 | SDIVT | O | Internally generated end fill bit |
| F0 | UDIVIT | O | Internally generated end fill bit |
| 7F | BCDBIN | I | Least significant bit |
| DF | BINEX3 | O | Shifted input to MQ register |

other arithmetic instructions

The 'ACT8832A supports two immediate arithmetic operations. ADDI and SUBI (Group 3) add or subtract a constant between the values of 0 and 15 from an operand on the S bus. The constant value is specified in bits A3-A0.

Twelve Group 4 instructions support serial division and multiplication. Signed, unsigned and mixed multiplication are implemented using three instructions: SMULI, which performs a signed times unsigned iteration; SMULT, which provides negative weighting of the sign bit of a negative multiplier in signed multiplication; and UMULI, which performs an unsigned multiplication iteration. Algorithms using these instructions are given in Tables 12, 13, and 14. These include: signed multiplication, which performs a two's complement multiplication; unsigned multiplication, which produces an unsigned times unsigned product; and mixed multiplication which multiplies a signed multiplicand by an unsigned multiplier to produce a signed result.

TABLE 12. SIGNED MULTIPLICATION ALGORITHM

| OP CODE | MNEMONIC | CLOCK CYCLES | INPUT S PORT | INPUT R PORT | OUTPUT Y PORT |
|---------|----------|------------------|--------------|--------------|----------------------------|
| E4 | LOADMQ | 1 | Multiplier | — | Multiplier |
| 60 | SMULI | N-1 [†] | Accumulator | Multiplicand | Partial product |
| 70 | SMULT | 1 | Accumulator | Multiplicand | Product (MSH) [‡] |

TABLE 13. UNSIGNED MULTIPLICATION ALGORITHM

| OP CODE | MNEMONIC | CLOCK CYCLES | INPUT S PORT | INPUT R PORT | OUTPUT Y PORT |
|---------|----------|------------------|--------------|--------------|----------------------------|
| E4 | LOADMQ | 1 | Multiplier | — | Multiplier |
| D0 | UMULI | N-1 [†] | Accumulator | Multiplicand | Partial product |
| D0 | UMULI | 1 | Accumulator | Multiplicand | Product (MSH) [‡] |

TABLE 14. MIXED MULTIPLICATION ALGORITHM

| OP CODE | MNEMONIC | CLOCK CYCLES | INPUT S PORT | INPUT R PORT | OUTPUT Y PORT |
|---------|----------|------------------|--------------|--------------|----------------------------|
| E4 | LOADMQ | 1 | Multiplier | — | Multiplier |
| 60 | SMULI | N-1 [†] | Accumulator | Multiplicand | Partial product |
| 60 | SMULI | 1 | Accumulator | Multiplicand | Product (MSH) [‡] |

[†]N = 8 for quad 8-bit mode, 16 for dual 16-bit mode, 32 for 32-bit mode.

[‡]The least significant half of the product is in the MQ register.

Instructions that support division include start, iterate, and terminate instructions for unsigned division routines (UDIVIS, UDIVI and UDIVIT); initialize, start, iterate, and terminate instructions for signed division routines (SDIVIN, SDIVIS, SDIVI, and SDIVIT); and correction instructions for these routines (DIVRF and SDIVQF). A Group 5 instruction, SDIVO, is available for optional overflow testing. Algorithms for signed and unsigned division are given in Tables 15 and 16. These use a nonrestoring technique to divide a 16 N-bit integer dividend by an 8 N-bit integer divisor to produce an 8 N-bit integer quotient and remainder, where N = 1 for quad 8-bit mode, N = 2 for dual 16-bit mode, and N = 4 for 32-bit mode.

TABLE 15. SIGNED DIVISION ALGORITHM

| OP CODE | MNEMONIC | CLOCK CYCLES | INPUT S PORT | INPUT R PORT | OUTPUT Y PORT |
|---------|----------|------------------|------------------------|--------------|------------------------|
| E4 | LOADMQ | 1 | Dividend (LSH) | — | Dividend (LSH) |
| 80 | SDIVIN | 1 | Dividend (MSH) | Divisor | Remainder (N) |
| AF | SDIVO | 1 | Remainder (N) | Divisor | Overflow Test Result |
| 90 | SDIVIS | 1 | Remainder (N) | Divisor | Remainder (N) |
| A0 | SDIVI | N-2 [†] | Remainder (N) | Divisor | Remainder (N) |
| E0 | SDIVIT | 1 | Remainder (N) | Divisor | Remainder [§] |
| 40 | DIVRF | 1 | Remainder [‡] | Divisor | Remainder [†] |
| 50 | SDIVQF | 1 | MQ register | Divisor | Quotient [#] |

[†]N = 8 for quad 8-bit mode, 16 for dual 16-bit mode, 32 for 32-bit mode.

[‡]The least significant half of the product is in the MQ register.

[§]Unfixed

[†]Fixed (corrected)

[#]The quotient is stored in the MQ register. Remainder can be output at the Y port or stored in the register file accumulator.

TABLE 16. UNSIGNED DIVISION ALGORITHM

| OP CODE | MNEMONIC | CLOCK CYCLES | INPUT S PORT | INPUT R PORT | OUTPUT Y PORT |
|---------|----------|------------------|------------------------|--------------|------------------------|
| E4 | LOADMQ | 1 | Dividend (LSH) | — | Dividend (LSH) |
| B0 | UDIVIS | 1 | Dividend (MSH) | Divisor | Remainder (N) |
| C0 | UDIVI | N-1 [†] | Remainder (N) | Divisor | Remainder (N) |
| F0 | UDIVIT | 1 | Remainder (N) | Divisor | Remainder [‡] |
| 40 | DIVRF | 1 | Remainder [§] | Divisor | Remainder [§] |

[†]N = 8 in quad 8-bit mode, 16 in dual 16-bit mode, 32 in 32-bit mode

[‡]Unfixed

[§]Fixed (corrected)

data conversion instructions

Conversion of binary data to one's and two's complement can be implemented using the INCNR instruction (Group 1). SMTc (Group 3) permits conversion from two's complement representation to sign magnitude representation, or vice versa. Two's complement numbers can be converted to their positive value, using ABS (Group 3).

SNORM and DNORM (Group 4) provide for normalization of signed, single- and double-precision data. The operand is placed in the MQ register and shifted toward the most significant bit until the two most significant bits are of opposite value. Zeroes are shifted into the least significant bit, provided SIO is high or floating. (A low on SIO will shift a one into the least significant bit.) SNORM allows the number of shifts to be counted and stored in one of the register files to provide the exponent.

Data stored in binary-coded decimal form can be converted to binary using BCD BIN (Group 5). A routine for this conversion, given in Table 17, allows the user to convert an N-digit BCD number to a 4N-bit binary number in 4N + 8 clock cycles.

TABLE 17. BCD TO BINARY ALGORITHM

| OP CODE | MNEMONIC | CLOCK CYCLES | INPUT S PORT | INPUT R PORT | OUTPUT DESTINATION |
|-------------------------------|------------|--------------|--------------|--------------|---------------------|
| E4 | LOADMQ | 1 | BCD operand | — | MQ reg. |
| D2 | SUBR/MQSLC | 1 | Accumulator | Accumulator | Accumulator/MQ reg. |
| D2 | SUBR/MQSLC | 1 | Mask reg. | Mask reg. | Mask reg/MQ reg. |
| D1 | MQSLC | 2 | Don't care | Don't care | MQ reg. |
| 68 | ADDI (15) | 1 | Accumulator | Decimal 15 | Mask reg. |
| REPEAT N-1 TIMES [†] | | | | | |
| DA | AND/MQSLC | 1 | MQ reg. | Mask reg. | Interim reg/MQ reg. |
| D1 | ADD/MQSLC | 1 | Accumulator | Interim reg. | Interim reg/MQ reg. |
| 7F | BCDBIN | 1 | Interim reg. | Interim res. | Accumulator/MQ reg. |
| 7F | BCDBIN | 1 | Accumulator | Interim reg. | Accumulator/MQ reg. |
| END REPEAT | | | | | |
| FA | AND | 1 | MQ reg. | Mask reg. | Interim reg. |
| D1 | ADD MQSLC | 1 | Accumulator | Interim reg. | Accumulator |

[†]N = Number of BCD digits

BINEX3, EX3BC, and EX3C assist binary to excess-3 conversion. Using BINEX3, an N-bit binary number can be converted to an N/4- digit excess-3 number. For an algorithm, see Table 18.

SN74ACT8832A

32-BIT REGISTERED ALU

TABLE 18. BCD TO BINARY ALGORITHM

| OP CODE | MNEMONIC | CLOCK CYCLES | INPUT S PORT | INPUT R PORT | OUTPUT DESTINATION |
|-----------------------------|-------------|--------------|---------------|---------------|--------------------|
| E4 | LOADMQ | 1 | Binary number | — | MQ reg. |
| D2 | SUBR | 1 | Accumulator | Accumulator | Accumulator |
| D2 | SET1 (33)16 | 1 | Accumulator | Mask (33)16 | Accumulator |
| REPEAT N TIMES [†] | | | | | |
| DF | BINEX3 | 1 | Accumulator | Accumulator | Accumulator/MQ reg |
| 9F | EX3C | 1 | Accumulator | Internal data | Accumulator |
| END REPEAT | | | | | |

[†]N = Number of bits in binary number

bit and byte instructions

Four Group 3 instructions allow the user to test or set selected bits within a byte. SET1 and SET0 force selected bits of a selected byte (or bytes) to one and zero, respectively. TB1 and TB test selected bits of a selected byte (or bytes) for ones and zeros. The bits to be set or tested are specified by an 8-bit mask formed by the concatenation of register file address inputs C3-C0 and A3-A0. The register file addressed by B5-B0 is used as the destination operand for the set bit instructions. Register writes are inhibited for test bit instructions. Bytes to be operated on are selected by forcing SION low, where n represents the byte position and 0 represents the least significant byte. A high on the zero output pin signifies that the test data matches the mask; a low on the zero output indicates that the test has failed.

Individual bytes of data can also be manipulated using eight Group 3 byte arithmetic/logic instructions. Bytes can be added, subtracted, incremented, ORed, ANDed, and exclusive ORed. Like the bit instructions, bytes are selected by forcing SION low, but multiple bytes can be operated on only if they are adjacent to one another; at least one byte must be nonselected.

other instructions

SEL (Group 4) selects one of the ALU's two operands, S or R, depending on the state of the SSF pin. This instruction could be used in sort routines to select the larger or smaller of two operands by performing a subtraction and sending the status result to SSF. CRC (Group 4) is designed to verify serial binary data that has been transmitted over a channel using a cyclic redundancy check code. An algorithm using this instruction is given in Table 19.

TABLE 19. CRC ALGORITHM

| OP CODE | MNEMONIC | CLOCK CYCLES | INPUT S PORT | INPUT R PORT | OUTPUT DESTINATION |
|--------------------------------|----------|--------------|--------------------------|-------------------|--------------------|
| E4 | LOADMQ | 1 | Vector $c'(x)^{\dagger}$ | — | MQ reg. |
| F6 | INCR | 1 | — | Polynomial $g(x)$ | Poly reg. |
| F2 | SUBR | 1 | Accumulator | Accumulator | Accumulator |
| REPEAT n/8N TIMES [†] | | | | | |
| 00 | CRC | 1 | Accumulator | Poly reg. | Accumulator |
| E4 | LOADMQ | 1 | Vector $c'(x)^{\dagger}$ | — | MQ reg. |
| END REPEAT | | | | | |

[†]N = Number of bits in binary number
n = Length of the code vector

CLR forces the ALU output to zero and clears the internal BCD flip-flops used in excess-3 BCD operations. NOP forces the ALU output to zero, but does not affect the flip-flops.

configuration options

The 'ACT8832A can be configured to operate in 8-bit, 16-bit, or 32-bit modes, depending on the setting of the configuration mode selects (CF2-CF0). Table 4 shows the control inputs for the four operating modes. Selecting an operating configuration other than 32-bit mode affects ALU operation and status generation in several ways, depending on the mode selected.

masked 32-bit operation

Masked 32-bit operation is selected to reset to zero the 20 most significant bits of the R Mux input. The 12 least significant bits are unaffected by the mask. Only Group 1 and Group 2 instructions can be used in this operating configuration. Status generation is similar to unmasked 32-bit operating mode.

shift instructions

Shift instructions operate similarly in 8-bit, 16-bit, and 32-bit modes. The serial I/O ($\overline{\text{SIO}}3'$ - $\overline{\text{SIO}}0'$) pins are used to select end-fill bits or to shift bits in or out, depending on the operation being performed. Table 6 shows the $\overline{\text{SIO}}$ signals associated with each byte or word in the different modes, and Table 11 indicates the specific function performed by the $\overline{\text{SIO}}$ pins during shift, multiply, and divide operations.

Figures 6 and 7 present examples of logical right shifts in 16-bit and 8-bit configurations.

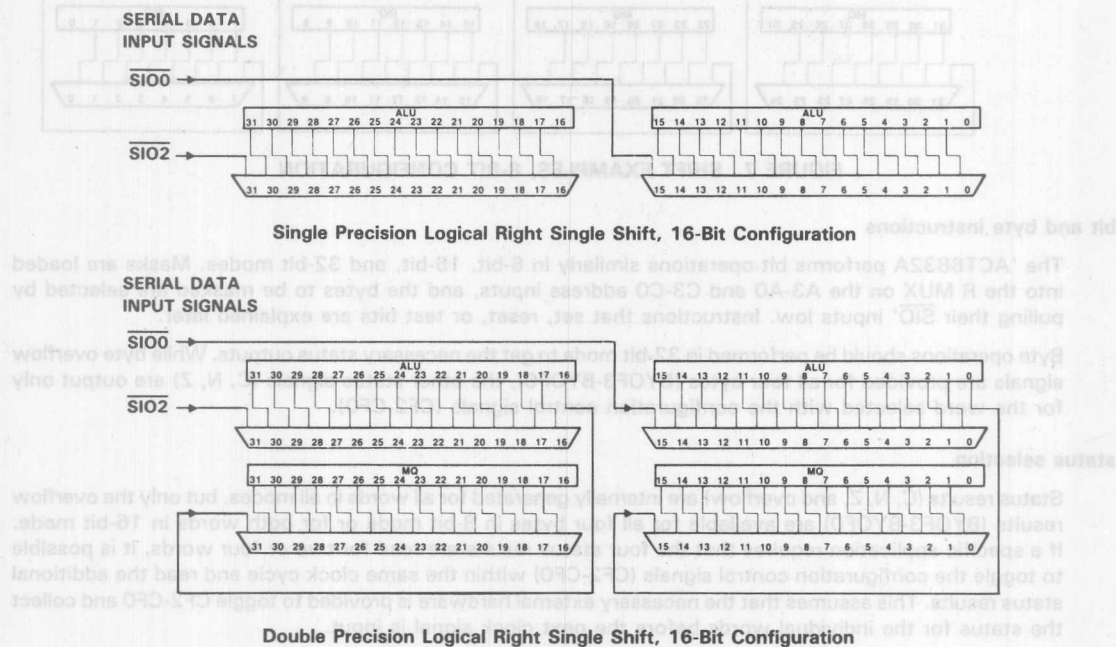


FIGURE 6. SHIFT EXAMPLES, 16-BIT CONFIGURATION

SN74ACT8832A **32-BIT REGISTERED ALU**

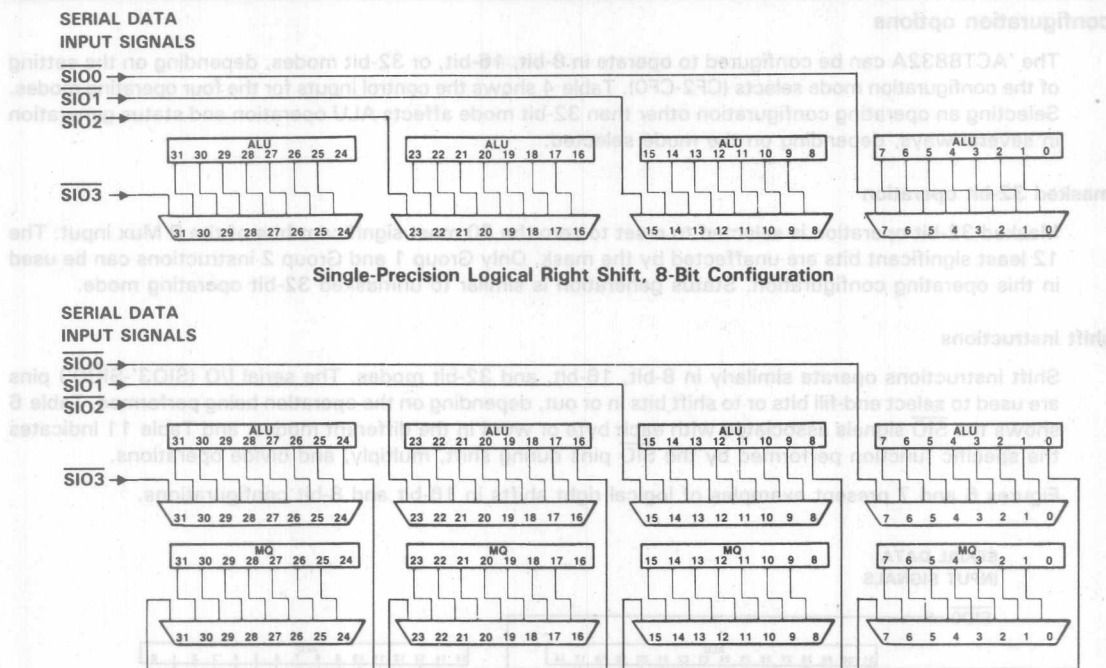


FIGURE 7. SHIFT EXAMPLES, 8-BIT CONFIGURATION

bit and byte instructions

The 'ACT8832A performs bit operations similarly in 8-bit, 16-bit, and 32-bit modes. Masks are loaded into the R MUX on the A3-A0 and C3-C0 address inputs, and the bytes to be masked are selected by pulling their SIO' inputs low. Instructions that set, reset, or test bits are explained later.

Byte operations should be performed in 32-bit mode to get the necessary status outputs. While byte overflow signals are provided for all four bytes (BYOF3-BYOF0), the other status signals (C, N, Z) are output only for the word selected with the configuration control signals (CF2-CF0).

status selection

Status results (C, N, Z, and overflow) are internally generated for all words in all modes, but only the overflow results (BYOF3-BYOF0) are available for all four bytes in 8-bit mode or for both words in 16-bit mode. If a specific application requires that the four status results are read for two or four words, it is possible to toggle the configuration control signals (CF2-CF0) within the same clock cycle and read the additional status results. This assumes that the necessary external hardware is provided to toggle CF2-CF0 and collect the status for the individual words before the next clock signal is input.

SN74ACT8832A 32-BIT REGISTERED ALU

absolute maximum ratings over operating free-air temperature range (unless otherwise noted)[†]

| | |
|--|----------------|
| Supply voltage, V_{CC} (see Note 1) | -0.5 V to 6 V |
| Input clamp current, I_{IK} ($V_I < 0$ or $V_I > V_{CC}$) | ± 20 mA |
| Output clamp current, I_{OK} ($V_O < 0$ or $V_O > V_{CC}$) | ± 50 mA |
| Continuous output current, I_O ($V_O = 0$ to V_{CC}) | ± 50 mA |
| Continuous current through V_{CC} or GND pins | ± 100 mA |
| Operating free-air temperature range | 0°C to 70°C |
| Storage temperature range | -65°C to 150°C |

[†]Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTE 1: All voltage levels are with respect to GND.

recommended operating conditions

| PARAMETER | | MIN | NOM | MAX | UNIT |
|-----------|------------------------------------|------|-----|----------|------|
| V_{CC} | Supply voltage | 4.75 | 5.0 | 5.25 | V |
| V_{IH} | High-level input voltage | 2 | | V_{CC} | V |
| V_{IL} | Low-level input voltage | 0 | | 0.8 | V |
| I_{OH} | High-level output current | | | -8 | mA |
| I_{OL} | Low-level output current | | | 8 | mA |
| V_I | Input voltage | 0 | | V_{CC} | V |
| V_O | Output voltage | 0 | | V_{CC} | V |
| dt/dv | Input transition rise or fall rate | 0 | | 15 | ns/V |
| T_A | Operating free-air temperature | 0 | | 70 | °C |

electrical characteristics over recommended ranges of supply voltage and free-air temperature (unless otherwise noted)

| PARAMETER | TEST CONDITIONS | V_{CC} | MIN | TYP [‡] | MAX | UNIT |
|---------------------------|---|----------|------|------------------|---------|---------|
| V_{OH} | $I_{OH} = -20 \mu A$ | 4.75 V | 4.6 | | | V |
| | | 5.25 V | 5.1 | | | |
| | $I_{OH} = -8 \text{ mA}$ | 4.75 V | 3.85 | 3.95 | | V |
| | | 5.25 V | 4.60 | 4.70 | | |
| V_{OL} | $I_{OL} = 20 \mu A$ | 4.75 V | | | 0.1 | V |
| | | 5.25 V | | | 0.1 | |
| | $I_{OL} = 8 \text{ mA}$ | 4.75 V | | 0.32 | 0.45 | |
| | | 5.25 V | | 0.32 | 0.45 | |
| I_I^{\S} | $V_I = V_{CC}$ or 0 | 5.25 V | | ± 0.1 | ± 1 | μA |
| $I_{IESIO3-0}$ | $V_I = V_{CC}$ or 0 | 5.25 V | | | ± 5 | mA |
| I_{CCQ} | $V_I = V_{CC}$ or 0 | 5.25 V | | 100 | 200 | μA |
| C_i | $V_I = V_{CC}$ or 0 | 5 V | | 10 | | pF |
| ΔI_{CC}^{\dagger} | One input at 3.4 V, other inputs at 0 or V_{CC} | 5.25 V | | | 1 | mA |

[‡]All typical values are at $V_{CC} = 5 \text{ V}$, $T_A = 25^\circ \text{C}$.

^{\S}For I/O ports, the parameter I_{OZH} and I_{OZL} include the offstate output current.

^{\dagger}This is the increase in supply current for each input that is at one of the specified TTL voltage levels rather than 0 V to V_{CC} .



SN74ACT8832A
32-BIT REGISTERED ALU

maximum switching characteristics over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)[†]

| PARAMETER | FROM (INPUT) | TO (OUTPUT) | | | | | | | | | | | UNIT |
|-----------------|----------------------------------|-------------|----|----|-----|---------|----|-----|------|----|-------|-------|------|
| | | Y | C | Z | SIO | PERRA/B | N | OVR | PA/B | PY | PERRY | MSERR | |
| t _{pd} | A5-A0,B5-B0 | 36 | 30 | 37 | 28 | | 30 | 37 | 16 | 37 | | | ns |
| | DA31-DA0,PA3-PA0 | 36 | 25 | 37 | 25 | 20 | 28 | 37 | | 37 | | | |
| | DB31-DB0,PB3-PB0 | | | | | | | | | | | | |
| | C _n | 30 | 22 | 31 | 24 | | 28 | 28 | | 32 | | | |
| | EA | 37 | 28 | 37 | 25 | | 31 | 37 | | 37 | | | |
| | EB1-EB0 | 37 | 28 | 37 | 25 | | 31 | 37 | | 37 | | | |
| | I7-I0 | 37 | 30 | 37 | 28 | | 32 | 37 | | 37 | | | |
| | CF2-CF0 | 37 | 30 | 37 | 28 | | 32 | 37 | | 37 | | | |
| | OE _B ,OE _A | | | | | | | | 15 | | | | |
| | OEY3-OEY0 | 20 | | | | | | | | 20 | | | |
| | SELMQ | 15 | | | | | | | | 20 | | | |
| | SIO3-SIO0 | 15 | | 25 | | | 25 | | | 27 | | | |
| | CLK | 21 | | | | | | | | 28 | | | |
| | CLK (w/MQ register feedback) | 37 | | | | | | | | 37 | | | |
| | RFCLK | 37 | 32 | 37 | 24 | | 32 | 37 | | 37 | | | |
| t _h | IESIO3-IESIO0 | 15 | | 25 | 15 | | 25 | | | 27 | | | ns |
| | SSF | 25 | | 30 | 22 | | 30 | 22 | | 30 | | | |
| | Y | | | | | | | | | | 15 | 15 | |

[†]See Parameter Measurement Information for load circuit and voltage waveforms.

timing requirements over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

file write setup

| PARAMETER | | MIN | MAX | UNIT |
|-----------------|---------------------------|-----|-----|------|
| t _{su} | C5-C0 | 4 | | ns |
| | DA/B32-DA/B0, PA/B3-PA/B0 | 7 | | |
| | I7-I0 | 13 | | |
| | OEY3-OEY0 | 7 | | |
| | Y31-Y0 | 4 | | |
| | WE3-WE0 | 4 | | |
| | SELRF(DA,DB,PA,PB) | 5 | | |
| | SELRF(Y) | 9 | | |
| | SIO | 10 | | |
| | SELMQ | 9 | | |
| t _h | IESIO3-IESIO0 | 10 | | ns |
| | ALL | 0 | | |

PARAMETER MEASUREMENT INFORMATION

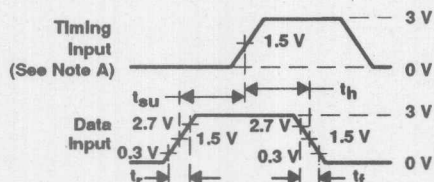
LOAD CIRCUIT PARAMETERS

| TIMING PARAMETERS | C_{LOAD}^{\dagger} (pF) | I_{OL} (mA) | I_{OH} (mA) | V_{LOAD} (V) |
|-------------------|------------------------------|------------------|------------------|-------------------|
| t_{en} | 50 | 8 | -8 | 0 |
| | | | | 3 |
| t_{dis} | 50 | 8 | -8 | 1.5 |
| | | | | 3 |
| t_{pd} | 50 | 8 | -8 | ‡ |

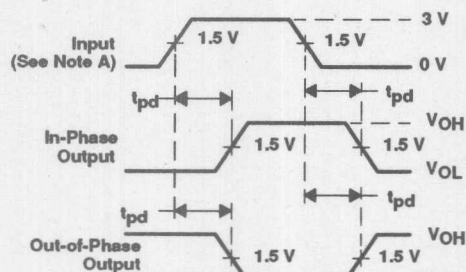
$\dagger C_{LOAD}$ includes probes and test fixture capacitance.

$\ddagger V_{LOAD} - V_{OL} = 50 \Omega$, where $V_{OL} = 0.6 V$, $I_{OL} = 8 mA$.

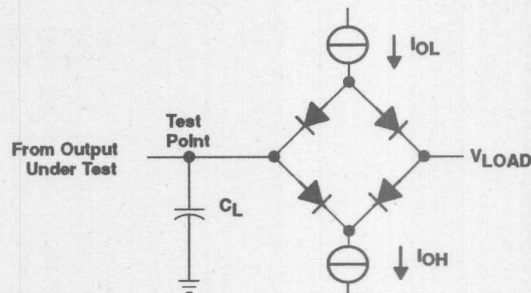
I_{OL}



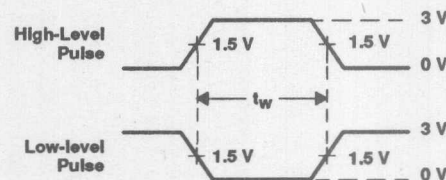
VOLTAGE WAVEFORMS
SETUP AND HOLD TIMES
INPUT RISE AND FALL TIMES



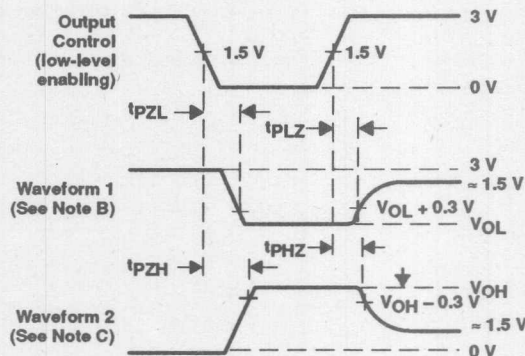
VOLTAGE WAVEFORMS
PROPAGATION DELAY TIMES



LOAD CIRCUIT



VOLTAGE WAVEFORMS
PULSE DURATION



VOLTAGE WAVEFORMS
ENABLE AND DISABLE TIMES, 3-STATE OUTPUTS

- Notes: A. Phase relationships between waveforms were chosen arbitrarily. All input pulses are supplied by pulse generators having the following characteristics: PRR = 1 MHz, $Z_0 = 50 \Omega$, $t_r \leq 6 ns$, $t_f \leq 6 ns$.
B. Waveform 1 is for an output with internal conditions such that the output is low except when disabled by the output control.
C. Waveform 2 is for an output with internal conditions such that the output is high except when disabled by the output control. For t_{PLZ} and t_{PHZ} , V_{OL} and V_{OH} are specified values.

FIGURE 8

Status Signals[†]

ZERO = 1 if result = 0
N = 1 if MSB of result = 1
 = 0 if MSB of result = 0
OVR = 1 if signed arithmetic overflow
C = 1 if carry-out = 1

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Add data in register 1 to data in register 10 with carry-in and store the unshifted result in register 1. Shift the contents of the MQ register one bit to the right, retaining the sign bit.

| Instr Code I7-I0 | Oprd Addr A5-A0 | Oprd Addr B5-B0 | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2- CF0 |
|------------------------|-----------------------|-----------------------|----------------------------|-----------------------|---------------------|-------------|-------------------|----|----|---------------|-----|---|-----|-------------|
| | | | | | SEL | WE3- WE0 | SELRF1- SELRF0 | OE | OE | OEY3- OEY0 | OES | | | |
| 1010 0001 | 00 0001 | 00 1010 | 0 00 | 00 0001 | 0 | 0000 | 10 | X | X | XXXX | 0 | 1 | 110 | |

Assume register file 1 holds 5608C618 (Hex), register file 10 holds 14007530 (Hex), and MQ register holds 98A99A0E (Hex).

Source 0101 0110 0000 1000 1100 0110 0001 1000 **R ← RF(1)**

Source 0001 0100 0000 0000 0111 0101 0111 0000 **S ← RF(10)**

Destination 0110 1010 0000 1001 0011 1011 0100 1001 **RF(1) ← R + S + Cn**

Source 1001 1000 1010 1001 1001 1010 0000 1110 **MQ shifter ← MQ register**

Destination 1100 1100 0101 0100 1100 1101 0000 0111 **MQ register ← MQ shifter**

FUNCTION

Passes the result of the ALU instruction specified in the upper nibble of the instruction field to Y MUX. Performs a right shift on MQ.

DESCRIPTION

The result of the arithmetic or logical operation specified in the lower nibble of the instruction field (I3-I0) is passed unshifted to Y MUX.

The contents of the MQ register are shifted one bit to the right. A zero is placed in the sign bit of the most significant byte unless the \overline{SIO} input for that byte is set to zero; this will force the sign bit to 1. Bit 0 of the least significant byte is dropped.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to the MQ register. If SSF is low, the MQ register will not be altered.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

| ALU Shifter | MQ Shifter |
|-------------|---------------|
| None | Logical Right |

Available Destination Operands (ALU Shifter)

| RF (C5-C0) | RF (B5-B0) | Y-Port |
|---------------|---------------|--------|
| Yes | No | Yes |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|-------------------|--|
| SSF | Yes | Passes shift result if high or floating; retains MQ without shift if low. |
| $\overline{SIO0}$ | Yes | Fills a zero in LSB of MQ shifter if high or floating; sets LSB to one if low. |
| $\overline{SIO1}$ | No | Inactive in 32-bit configuration; used in other configurations to select end-fill in LSBs. |
| $\overline{SIO2}$ | No | |
| $\overline{SIO3}$ | No | |
| Cn | No | Affects arithmetic operation programmed in bits I3-I0 of instruction field. |

Status Signals[†]

ZERO = 1 if result = 0
N = 1 if MSB of result = 1
 = 0 if MSB of result = 0
OVR = 1 if signed arithmetic overflow
C = 1 if carry-out = 1

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Add data in register 1 to data on the DB bus with carry-in and store the unshifted result in register 1. Shift the contents of the MQ register one bit to the left.

| Instr Code I7-I0 | Oprd Addr A5-A0 | Oprd Addr B5-B0 | Oprd Sel EB1- EA EB0 | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2- CF0 |
|------------------------|-----------------------|-----------------------|----------------------------|-----------------------|---------------------|-------------|-------------------|-------------|-------------|-------------|-------------|-------------|-----|-------------|
| | | | | | SELMO | WE3- WE0 | SELRF1- SELRF0 | OE3- OE0 | OE2- OE1 | OE1- OE0 | OE0- OE0 | OE0- OE0 | | |
| 1011 0001 | 00 0001 | XX XXXX | 0 10 | 00 0001 | 0 | 0000 | 10 | X | X | XXXX | 0 | 1 | 110 | |

Assume register file 1 holds 5608C618 (Hex), DB bus holds 14007530 (Hex), and MQ register holds 98A99A0E (Hex).

Source 0101 0110 0000 1000 1100 0110 0001 1000 $R \leftarrow \text{RF}(1)$

Source 0001 0100 0000 0000 0111 0101 0011 0000 $S \leftarrow \text{DB bus}$

Destination 0110 1010 0000 1001 0011 1011 0100 1001 $\text{RF}(1) \leftarrow R + S + \text{Cn}$

Source 1001 1000 1010 1001 1001 1010 0000 1110 MQ shifter \leftarrow MQ register

Destination 0100 1100 0101 0100 1100 1101 0000 0111 MQ register \leftarrow MQ shifter

FUNCTION

Evaluates the logical expression R NAND S.

DESCRIPTION

Data on the R bus is Nanded with data on the S bus. The result appears at the ALU and MQ shifters.

*The result of this instruction can be shifted in the same microcycle by specifying a shift instruction in the upper nibble (I7-I4) of the instruction field. The result may also be passed without shift. Possible instructions are listed in Table 15.

Available R Bus Source Operands

| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
|---------------|----------------|---------|------------------------------|
| Yes | No | Yes | No |

Available S Bus Source Operands

| RF (B5-B0) | DB-Port | MQ Register |
|---------------|---------|----------------|
| Yes | Yes | Yes |

Available Destination Operands

| RF (C5-C0) | RF (B5-B0) | Y-Port | ALU Shifter | MQ Shifter |
|---------------|---------------|--------|----------------|---------------|
| Yes | No | Yes | Yes | Yes |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|----------------------|--|
| SSF | No | Affect shift instructions programmed in bits I7-I4 of instruction field. |
| $\overline{SIO0}$ | No | |
| $\overline{SIO1}$ | No | |
| $\overline{SIO2}$ | No | |
| $\overline{SIO3}$ | No | |
| Cn | | Inactive |

Status Signals[†]

| | |
|------|-------------------|
| ZERO | = 1 if result = 0 |
| N | = 1 if MSB = 1 |
| OVR | = 0 |
| C | = 0 |

[†]C is ALU carry out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Logically NAND the contents of register 3 and register 5, and store the result in register 5.

| Instr Code I7-I0 | Oprd Addr A5-A0 | Oprd Addr B5-B0 | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | Cn | CF2- CF0 |
|------------------------|-----------------------|-----------------------|----------------------------|-----------------------|---------------------|-------------|-------------------|-----|-----|---------------|-----|----|-------------|
| | | | | | SELMO | WE3- WE0 | SELRF1- SELRF0 | OEA | OEB | OEY3- OEY0 | OES | | |
| 1111 1100 | 00 0011 | 00 0101 | 0 00 | 00 0101 | 0 | 0000 | 10 | X | X | XXXX | 0 | X | 110 |

Assume register file 1 holds 60F6D840 (Hex) and register file 5 holds 13F6D377 (Hex).

Source

0110 0000 1111 0110 1101 1000 0100 0000

 R ← RF(3)

Source

0001 0011 1111 0110 1101 0011 0111 0111

 S ← RF(5)

Destination

1111 1111 0000 1001 0010 1111 1011 1111

 RF(5) ← R NAND S

| | |
|------|-----|
| ZERO | = 1 |
| N | = 0 |
| OVR | = 0 |
| C | = 0 |

NOP**No Operation****F F****FUNCTION**

Forces ALU output to zero.

DESCRIPTION

This instruction forces the ALU output to zero. The BCD flip-flops retain their old value. Note that the clear instruction (CLR) forces the ALU output to zero and clears the BCD flip-flops.

Available R Bus Source Operands

| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
|---------------|----------------|---------|------------------------------|
| No | No | No | No |

Available S Bus Source Operands

| RF (B5-B0) | DB-Port | MQ Register |
|---------------|---------|----------------|
| No | No | No |

Available Destination Operands

| RF (C5-C0) | RF (B5-B0) | Y-Port |
|---------------|---------------|--------|
| Yes | No | Yes |

Shift Operations

| ALU | MQ |
|------|------|
| None | None |

Status Signals

| |
|----------|
| ZERO = 1 |
| N = 0 |
| OVR = 0 |
| C = 0 |

EXAMPLE (assumes a 32-bit configuration)

Clear register 12.

| Instr Code I7-I0 | Oprd Addr A5-A0 | Oprd Addr B5-B0 | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2 CF0 |
|------------------------|-----------------------|-----------------------|----------------------------|-----------------------|---------------------|-------------|-----------------|-------------|---------------|------|---|---|-----|------------|
| | | | | | SELMQ | WE3- WE0 | SELR1- SELR0 | OE3- OE0 | OEY3- OEY0 | OES | | | | |
| 1111 1111 | XX XXXX | XX XXXX | X XX | 00 1100 | 0 | 0000 | 10 | X | X | XXXX | 0 | X | 110 | |

Destination 0000 0000 0000 0000 0000 0000 0000 0000 RF(12) ← 0

FUNCTION

Evaluates the logical expression R NOR S.

DESCRIPTION

Data on the R bus is NORed with data on the S bus. The result appears at the ALU and MQ shifters.

*The result of this instruction can be shifted in the same microcycle by specifying a shift instruction in the upper nibble (I7-I4) of the instruction field. The result may also be passed without shift. Possible instructions are listed in Table 15.

Available R Bus Source Operands

| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
|---------------|----------------|---------|------------------------------|
| Yes | No | Yes | No |

Available S Bus Source Operands

| RF (B5-B0) | DB-Port | MQ Register |
|---------------|---------|----------------|
| Yes | Yes | Yes |

Available Destination Operands

| RF (C5-C0) | RF (B5-B0) | Y-Port | ALU Shifter | MQ Shifter |
|---------------|---------------|--------|----------------|---------------|
| Yes | No | Yes | Yes | Yes |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|----------------------|--|
| SSF | No | Affect shift instructions programmed in bits I7-I4 of instruction field. |
| $\overline{SIO0}$ | No | |
| $\overline{SIO1}$ | No | |
| $\overline{SIO2}$ | No | |
| $\overline{SIO3}$ | No | |
| Cn | No | Inactive |

Status Signals†

ZERO = 1 if result = 0
 N = 1 if MSB = 1
 OVR = 0
 C = 0

†C is ALU carry out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Logically NOR the contents of register 3 and register 5, and store the result in register 5.

| Instr Code I7-I0 | Oprd Addr A5-A0 | Oprd Addr B5-B0 | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | Cn | CF2- CF0 |
|------------------------|-----------------------|-----------------------|----------------------------|-----------------------|------------------------|--------|--------------|-------------|-------------|-------------|----|-------------|
| | | | | | WE3- SELRF1- WEO | SELRF0 | OE3- OEY0 | OE3- OES | OE3- OES | OE3- OES | | |
| 1111 1011 | 00 0011 | 00 0101 | 0 00 | 00 0101 | 0 | 0000 | 10 | X | X | XXXX | 0 | X 110 |

Assume register file 3 holds 60F6D840 (Hex) and register file 5 holds 13F6D377 (Hex).

Source 0110 0000 1111 0110 1101 1000 0100 0000 R ← RF(3)

Source 0001 0011 1111 0110 1101 0011 0111 0111 S ← RF(5)

Destination 1000 1100 0000 1001 0010 0100 1000 1000 RF(5) ← R NOR S

FUNCTION

Evaluates the logical expression R OR S.

DESCRIPTION

Data on the R bus is ORed with data on the S bus. The result appears at the ALU and MQ shifters.

*The result of this instruction can be shifted in the same microcycle by specifying a shift instruction in the upper nibble (I7-I4) of the instruction field. The result may also be passed without shift. Possible instructions are listed in Table 15.

Available R Bus Source Operands

| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
|---------------|----------------|---------|------------------------------|
| Yes | No | Yes | No |

Available S Bus Source Operands

| RF (B5-B0) | DB-Port | MQ Register |
|---------------|---------|----------------|
| Yes | Yes | Yes |

Available Destination Operands

| RF (C5-C0) | RF (B5-B0) | Y-Port | ALU Shifter | MQ Shifter |
|---------------|---------------|--------|----------------|---------------|
| Yes | No | Yes | Yes | Yes |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|----------------------|--|
| SSF | No | Affect shift instructions programmed in bits I7-I4 of instruction field. |
| $\overline{SIO0}$ | No | |
| $\overline{SIO1}$ | No | |
| $\overline{SIO2}$ | No | |
| $\overline{SIO3}$ | No | |
| Cn | No | Inactive |

Status Signals†

ZERO = 1 if result = 0
 N = 1 if MSB = 1
 OVR = 0
 C = 0

†C is ALU carry out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Logically OR the contents of register 5 and register 3, and store the result in register 3.

| Instr Code 17-10 | Oprd Addr A5-A0 | Oprd Addr B5-B0 | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | Cn | CF2 CF0 |
|------------------------|-----------------------|-----------------------|----------------------------|-----------------------|---------------------|----------------|--------|-----|-----|--------------|-----|----|------------|
| | | | | | WE3- SELMO | SELRF1- WE0 | SELRF0 | OEA | OEB | OY3- OEY0 | OES | | |
| 1111 1011 | 00 0101 | 00 0011 | 0 00 | 00 0011 | 0 | 0000 | 10 | X | X | XXXX | 0 | X | 110 |

Assume register file 5 holds 60F6D840 (Hex) and register file 3 holds 13F6D377 (Hex).

Source 0110 0000 1111 0110 1101 1000 0100 0000 R ← RF(5)

Source 0001 0011 1111 0110 1101 0011 0111 0111 S ← RF(3)

Destination 0111 0011 1111 0110 1101 1011 0111 0111 RF(3) ← R OR S

PASS

Pass (Y ← F)

F *

FUNCTION

Passes the result of the ALU instruction specified in the lower nibble of the instruction field to Y MUX.

DESCRIPTION

The result of the arithmetic or logical operation specified in the lower nibble of the instruction field (I3-I0) is passed unshifted to Y MUX.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Available Destination Operands

| RF (C5-C0) | RF (B5-B0) | Y-Port | ALU Shifter | MQ Shifter |
|---------------|---------------|--------|----------------|---------------|
| Yes | No | Yes | None | None |

Control/Data Signals

| Signal | User Programmable | Use |
|--------|----------------------|--|
| SSF | No | Inactive |
| SIO0 | No | Inactive |
| SIO1 | No | Inactive |
| SIO2 | No | Inactive |
| SIO3 | No | Inactive |
| Cn | No | Affects arithmetic operation specified in bits I3-I0 of instruction field. |

Status Signals†

| | |
|------|--|
| ZERO | = 1 if result = 0 |
| N | = 1 if MSB of result = 1 = 0 if MSB of result = 0 |
| OVR | = 1 if signed arithmetic overflow |
| C | = 1 if carry-out condition |

†C is ALU carry out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Add data in register 1 to data on the DB bus with carry-in and store the unshifted result in register 10.

| Instr Code I7-I0 | Oprd Addr A5-A0 | Oprd Addr B5-B0 | Oprd Sel EB1- EA EB0 | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2- CF0 |
|------------------------|-----------------------|-----------------------|----------------------------|-----------------------|---------------------|-------------|-------------------|-------------|-------------|-------------|-------------|-------------|-----|-------------|
| | | | | | SELMQ | WE3- WE0 | SELRF1- SELRF0 | OE3- OE0 | OE1- OE0 | OE2- OE0 | OE3- OE0 | OE1- OE0 | | |
| 1111 0001 | 00 0001 | XX XXXX | 0 10 | 00 1010 | 0 | 0000 | 10 | X | X | XXXX | 0 | 1 | 110 | |

Assume register file 3 holds 9308C618 (Hex) and DB bus holds 24007530 (Hex).

| | | |
|-------------|---|--------------------------------|
| Source | 1001 0011 0000 1000 1100 0110 0001 1000 | $R \leftarrow RF(1)$ |
| Source | 0010 0100 0000 0000 0111 0101 0011 0000 | $S \leftarrow DB\ bus$ |
| Destination | 1011 0111 0000 1001 0011 1011 0100 1001 | $RF(10) \leftarrow R + S + Cn$ |

| | | | |
|------------------|-----|-----|-----|
| RF (A5-A0) Immed | Yes | Yes | Yes |
| DA-Port | No | Yes | No |
| Mask | No | Yes | No |
| C3-C0 | No | Yes | No |

| | | | |
|---------------------|-----|-----|----|
| RF (B5-B0) Register | Yes | Yes | No |
| DB-Port | No | Yes | No |
| MQ | No | Yes | No |

| | | |
|-----|------|------|
| ALU | Left | Left |
| MQ | Left | Left |

| | | | |
|--------------------|-----|-----|-----|
| RF (C3-C0) (B5-B0) | Yes | No | Yes |
| Y-Port | No | Yes | Yes |

FUNCTION

Performs one of N-2 iterations of nonrestoring signed division by a test subtraction of the N-bit divisor from the 2N-bit dividend. An algorithm using this instruction is given in the "Other Arithmetic Instructions" section.

DESCRIPTION

SDIVI performs a test subtraction of the divisor from the dividend to generate a quotient bit. The test subtraction passes if the remainder is positive and fails if negative. If it fails, the remainder will be corrected during the next instruction.

SDIVI checks the pass/fail result of the test subtraction from the previous instruction, and evaluates

$$\begin{aligned} F &\leftarrow R + S && \text{if the test fails} \\ F &\leftarrow R' + S + C_n && \text{if the test passes} \end{aligned}$$

A double precision left shift is performed; bit 7 of the most significant byte of the MQ shifter is transferred to bit 0 of the least significant byte of the ALU shifter. Bit 7 of the most significant byte of the ALU shifter is lost. The unfixed quotient bit is circulated into the least significant bit of the MQ shifter.

The R bus must be loaded with the divisor, the S bus with the most significant half of the result of the previous instruction (SDIVI during iteration or SDIVIS at the beginning of iteration). The least significant half of the previous result is in the MQ register. Carry-in should be programmed high. Overflow occurring during SDIVI is reported to OVR at the end of the signed divide routine (after SDIVQF).

Available R Bus Source Operands

| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
|---------------|----------------|---------|------------------------------|
| Yes | No | Yes | No |

Recommended S Bus Source Operands

| RF (B5-B0) | DB-Port | MQ Register |
|---------------|---------|----------------|
| Yes | Yes | No |

Recommended Destination Operands

| RF (C5-C0) | RF (B5-B0) | Y-Port |
|---------------|---------------|--------|
| Yes | No | Yes |

Shift Operations

| ALU | MQ |
|------|------|
| Left | Left |

Control/Data Signals

| Signal | User Programmable | Use |
|--------------------------|-------------------|--|
| SSF | No | Inactive |
| $\overline{\text{SI00}}$ | No | Pass internally generated end-fill bits. |
| $\overline{\text{SI01}}$ | No | |
| $\overline{\text{SI02}}$ | No | |
| $\overline{\text{SI03}}$ | No | |
| Cn | Yes | Should be programmed high |

Status Signals

| | |
|------|--------------------------------|
| ZERO | = 1 if intermediate result = 0 |
| N | = 0 |
| OVR | = 0 |
| C | = 1 if carry-out |

FUNCTION

Initializes 'ACT8832 for nonrestoring signed division by shifting the dividend left and internally preserving the sign bit. An algorithm using this instruction is given in the "Other Arithmetic Instructions section.

DESCRIPTION

This instruction prepares for signed divide iteration operations by shifting the dividend and storing the sign for future use.

The preceding instruction should load the MQ register with the least significant half of the dividend. During SDIVIN, the S bus should be loaded with the most significant half of the dividend, and the R bus with the divisor. Y-output should be written back to the register file for use in the next instruction.

A double precision logical left shift is performed; bit 7 of the most significant byte of the MQ shifter is transferred to bit 0 of the least significant byte of the ALU shifter. Bit 7 of the most significant byte of the ALU shifter is lost. The unfixed quotient sign bit is shifted into the least significant bit of the MQ shifter.

Available R Bus Source Operands

| | | | |
|---------------|----------------|---------|------------------------------|
| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
| Yes | No | Yes | No |

Recommended S Bus Source Operands

| | | |
|---------------|---------|----------------|
| RF (B5-B0) | DB-Port | MQ Register |
| Yes | Yes | No |

Recommended Destination Operands

| | | |
|---------------|---------------|--------|
| RF (C5-C0) | RF (B5-B0) | Y-Port |
| Yes | No | Yes |

Shift Operations

| | |
|------|------|
| ALU | MQ |
| Left | Left |

Control/Data Signals

| Signal | User Programmable | Use |
|--------------------------|-------------------|--|
| SSF | No | Inactive |
| $\overline{\text{SIO0}}$ | No | Pass internally generated end-fill bits. |
| $\overline{\text{SIO1}}$ | No | |
| $\overline{\text{SIO2}}$ | No | |
| $\overline{\text{SIO3}}$ | No | |
| Cn | No | Inactive |

Status Signals

| |
|-------------------------|
| ZERO = 1 if divisor = 0 |
| N = 0 |
| OVR = 0 |
| Cn = 0 |

FUNCTION

Computes the first quotient bit of nonrestoring signed division. An algorithm using this instruction is given in the "Other Arithmetic Instructions" section..

DESCRIPTION

SDIVIS computes the first quotient bit during nonrestoring signed division by subtracting the divisor from the dividend, which was left-shifted during the prior SDIVIN instruction. The resulting remainder due to subtraction may be negative. If so, the subsequent SDIVI instruction will restore the remainder during the next subtraction.

The R bus must be loaded with the divisor and the S bus with the most significant half of the remainder. The result on the Y bus should be loaded back into the register file for use in the next instruction. The least significant half of the remainder is in the MQ register. Carry-in should be programmed high.

A double precision left shift is performed; bit 7 of the most significant byte of the MQ shifter is transferred to bit 0 of the least significant byte of the ALU shifter. Bit 7 of the most significant byte of the ALU shifter is lost. The unfixed quotient bit is circulated into the least significant bit of the MQ shifter.

Overflow occurring during SDIVIS is reported to OVR at the end of the signed division routine (after SDIVQF).

Available R Bus Source Operands

| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
|---------------|----------------|---------|------------------------------|
| Yes | No | Yes | No |

Recommended S Bus Source Operands

| RF (B5-B0) | DB-Port | MQ Register |
|---------------|---------|----------------|
| Yes | Yes | No |

Recommended Destination Operands

| RF (C5-C0) | RF (B5-B0) | Y-Port |
|---------------|---------------|--------|
| Yes | No | Yes |

Shift Operations

| ALU | MQ |
|------|------|
| Left | Left |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|-------------------|--|
| SSF | No | Inactive |
| $\overline{SIO0}$ | No | Pass internally generated end-fill bits. |
| $\overline{SIO1}$ | No | |
| $\overline{SIO2}$ | No | |
| $\overline{SIO3}$ | No | |
| Cn | Yes | Should be programmed high. |

Status Signals

| | |
|------|--------------------------------|
| ZERO | = 1 if intermediate result = 0 |
| N | = 0 |
| OVR | = 0 |
| C | = 1 if carry-out |

FUNCTION

Solves the final quotient bit during nonrestoring signed division. An algorithm using this instruction is given in the "Other Arithmetic Instructions" section.

DESCRIPTION

SDIVIT performs the final subtraction of the divisor from the remainder during nonrestoring signed division. SDIVIT is preceded by N-2 iterations of SDIVI, where N is the number of bits in the dividend.

The R bus must be loaded with the divisor, and the S bus must be loaded with the most significant half of the result of the last SDIVI instruction. The least significant half lies in the MQ register. The Y bus result must be loaded back into the register file for use in the subsequent DIVRF instruction. Carry-in should be programmed high.

SDIVIT checks the pass/fail result of the previous instruction's test subtraction and evaluates;

$$\begin{aligned} Y &\leftarrow R + S && \text{if the test fails} \\ Y &\leftarrow R' + S + C_n && \text{if the test passes} \end{aligned}$$

The contents of the MQ register are shifted one bit to the left; the unfixed quotient bit is circulated into the least significant bit.

Overflow during this instruction is reported to OVR at the end of the signed division routine (after SDIVQF).

Available R Bus Source Operands

| | | | |
|---------------|----------------|---------|------------------------------|
| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
| Yes | No | Yes | No |

Recommended S Bus Source Operands

| | | |
|---------------|---------|----------------|
| RF (B5-B0) | DB-Port | MQ Register |
| Yes | Yes | No |

Recommended Destination Operands

| | | |
|---------------|---------------|--------|
| RF (C5-C0) | RF (B5-B0) | Y-Port |
| Yes | No | Yes |

Shift Operations

| | |
|------|------|
| ALU | MQ |
| Left | Left |

Control/Data Signals

| Signal | User Programmable | Use |
|--------------------------|-------------------|--|
| SSF | No | Inactive |
| $\overline{\text{SIO0}}$ | No | Pass internally generated end-fill bits. |
| $\overline{\text{SIO1}}$ | No | |
| $\overline{\text{SIO2}}$ | No | |
| $\overline{\text{SIO3}}$ | No | |
| Cn | Yes | Should be programmed high |

Status Signals

| | |
|------|--------------------------------|
| ZERO | = 1 if intermediate result = 0 |
| N | = 0 |
| OVR | = 0 |
| C | = 1 if carry-out |

FUNCTION

Tests for overflow during nonrestoring signed division. An algorithm using this instruction is given in the "Other Arithmetic Instructions section.

DESCRIPTION

This instruction performs an initial test subtraction of the divisor from the dividend. If overflow is detected, it is preserved internally and reported at the end of the divide routine (after SDIVQF). If overflow status is ignored, the SDIVO instruction may be omitted.

The divisor must be loaded onto the R bus; the most significant half of the previous SDIVIN result must be loaded onto the S bus. The least significant half is in the MQ register.

The result on the Y bus should not be stored back into the register file; WE' should be programmed high.

Carry-in should also be programmed high.

Available R Bus Source Operands

| | | | |
|---------------|----------------|---------|------------------------------|
| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
| Yes | No | Yes | No |

Recommended S Bus Source Operands

| | | |
|---------------|---------|----------------|
| RF (B5-B0) | DB-Port | MQ Register |
| Yes | Yes | No |

Recommended Destination Operands

| | | |
|---------------|---------------|--------|
| RF (C5-C0) | RF (B5-B0) | Y-Port |
| Yes | No | Yes |

Shift Operations

| | |
|------|------|
| ALU | MQ |
| None | None |

Control/Data Signals

| Signal | User Programmable | Use |
|--------------------------|-------------------|---------------------------|
| SSF | No | Inactive |
| $\overline{\text{SIO0}}$ | No | Inactive |
| $\overline{\text{SIO1}}$ | No | Inactive |
| $\overline{\text{SIO2}}$ | No | Inactive |
| $\overline{\text{SIO3}}$ | No | Inactive |
| Cn | Yes | Should be programmed high |

Status Signals

| | |
|------|--------------------|
| ZERO | = 1 if divisor = 0 |
| N | = 0 |
| OVR | = 0 |
| C | = 1 if carry-out |

FUNCTION

Tests the quotient result after nonrestoring signed division and corrects it if necessary. An algorithm using this instruction is given in the "Other Arithmetic Instructions" section.

DESCRIPTION

SDIVQF is the final instruction required to compute the quotient of a 2N-bit dividend by an N-bit divisor. It corrects the quotient if the signs of the divisor and dividend are different and the remainder is nonzero.

The fix is implemented by incrementing S:

$Y \leftarrow S + 1$ if a fix is required
 $Y \leftarrow S + 0$ if no fix is required

The R bus must be loaded with the divisor, and the S bus with the most significant half of the result of the preceding DIVRF instruction. The least significant half is in the MQ register.

Available R Bus Source Operands

| | | | |
|---------------|----------------|---------|------------------------------|
| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
| Yes | No | Yes | No |

Recommended S Bus Source Operands

| | | |
|---------------|---------|----------------|
| RF (B5-B0) | DB-Port | MQ Register |
| Yes | Yes | No |

Recommended Destination Operands

| | | |
|---------------|---------------|--------|
| RF (C5-C0) | RF (B5-B0) | Y-Port |
| Yes | No | Yes |

Shift Operations

| | |
|------|------|
| ALU | MQ |
| None | None |

Control/Data Signals

| Signal | User Programmable | Use |
|--------------------------|-------------------|---------------------------|
| SSF | No | Inactive |
| $\overline{\text{SIO0}}$ | No | Inactive |
| $\overline{\text{SIO1}}$ | No | Inactive |
| $\overline{\text{SIO2}}$ | No | Inactive |
| $\overline{\text{SIO3}}$ | No | Inactive |
| Cn | Yes | Should be programmed high |

Status Signals

| | |
|------|-----------------------------|
| ZERO | = 1 if quotient = 0 |
| N | = 1 if sign of quotient + 1 |
| | = 0 if sign of quotient + 0 |
| OVR | = 1 if divide overflow |
| C | = 1 if carry-out |

FUNCTION

Selects S if SSF is high; otherwise selects R.

DESCRIPTION

Data on the S bus is passed to Y if SSF is programmed high or floating; data on the R bus is passed without carry to Y if SSF is programmed low.

Available R Bus Source Operands

| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
|---------------|----------------|---------|------------------------------|
| Yes | No | Yes | No |

Available S Bus Source Operands (MSH)

| RF (B5-B0) | DB-Port | MQ Register |
|---------------|---------|----------------|
| Yes | Yes | Yes |

Available Destination Operands

| RF (C5-C0) | RF (B5-B0) | Y-Port |
|---------------|---------------|--------|
| Yes | No | Yes |

Shift Operations

| ALU | MQ |
|------|------|
| None | None |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|----------------------|------------------------------|
| SSF | Yes | Selects S if high, R if low. |
| $\overline{SIO0}$ | No | Inactive |
| $\overline{SIO1}$ | No | Inactive |
| $\overline{SIO2}$ | No | Inactive |
| $\overline{SIO3}$ | No | Inactive |
| Cn | No | Inactive |

Status Signals

ZERO = 1 if result = 0

N = 1 if MSB = 1

OVR = 0

C = 0

EXAMPLE (assumes a 32-bit configuration)

Compare the two's complement numbers in registers 1 and 3 and store the larger in register 5.

1. Subtract (SUBS) data in register 3 from data in register 1 and pass the result to the Y bus.
2. Perform Select S/R instruction and pass result to register 5.

[This example assumes the SSF is set by the negative status (N) from the previous instruction].

| Instr Code I7-I0 | Oprd Addr A5-A0 | Oprd Addr B5-B0 | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2- CF0 |
|------------------------|-----------------------|-----------------------|----------------------------|-----------------------|---------------------|-------------|-------------------|-------------|---------------|------|---|---|-----|-------------|
| | | | | | SELMQ | WE3- WE0 | SELRF1- SELRF0 | OE3- OE0 | OEY3- OEY0 | OES | | | | |
| 1111 0011 | 00 0001 | 00 0011 | 0 00 | XX XXXX | 0 | XXXX | XX | X | X | 0000 | 0 | 1 | 110 | |
| 0001 0000 | 00 0001 | 00 0011 | 0 00 | 00 0101 | 0 | 0000 | 10 | X | X | XXXX | 0 | 0 | 110 | |

Assume register file 1 holds 008497D0 (Hex) and register file 3 holds 01C35250 (Hex).

Instruction Cycle 1

Source 0000 0000 1000 0100 1001 0111 1101 0000 $R \leftarrow RF(1)$

Source 0000 0001 1100 0011 0101 0010 0101 0000 $S \leftarrow RF(3)$

Destination 1111 1110 1100 0001 0100 0101 1000 0000 $Y \text{ bus} \leftarrow R + S' + Cn$

1 $N \leftarrow 1$

Instruction Cycle 2

Source 0000 0000 1000 0100 1001 0111 1101 0000 $R \leftarrow RF(1)$

1 $SSF \leftarrow 1$

Source 0000 0001 1100 0011 0101 0010 0101 0000 $S \leftarrow RF(3)$

Destination 0000 0001 1100 0011 0101 0010 0101 0000 $RF(5) \leftarrow S$

FUNCTION

Resets bits in selected bytes of S-bus data using mask in C3-C0::A3-A0.

DESCRIPTION

The register addressed by B5-B0 is both the source and destination for this instruction. The source word is passed on the S bus to the ALU, where it is compared to an 8-bit mask, consisting of a concatenation of the C3-C0 and A3-A0 address ports (C3-C0::A3-A0). The mask is input via the R bus. All bits in the source word that are in the same bit position as ones in the mask are reset. Bytes with their $\overline{\text{SIO}}$ inputs programmed low perform the Reset Bit instruction. Bytes with their $\overline{\text{SIO}}$ inputs programmed high or floating pass S unaltered.

Available R Bus Source Operands

| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
|---------------|----------------|---------|------------------------------|
| No | No | No | Yes |

Available S Bus Source Operands (MSH)

| RF (B5-B0) | DB-Port | MQ Register |
|---------------|---------|----------------|
| Yes | Yes | Yes |

Available Destination Operands

| RF (C5-C0) | RF (B5-B0) | Y-Port |
|---------------|---------------|--------|
| No | Yes | Yes |

Shift Operations

| ALU | MQ |
|------|------|
| None | None |

Control/Data Signals

| Signal | User Programmable | Use |
|--------------------------|-------------------|-------------|
| SSF | No | Inactive |
| $\overline{\text{SIO0}}$ | No | Byte-select |
| $\overline{\text{SIO1}}$ | No | Byte-select |
| $\overline{\text{SIO2}}$ | No | Byte-select |
| $\overline{\text{SIO3}}$ | No | Byte-select |
| Cn | No | Inactive |

Status Signals

ZERO = 1 if result (selected bytes) = 0

N = 0

OVR = 0

C = 0

EXAMPLE (assumes a 32-bit configuration)

Set bits 3-0 of bytes 1 and 2 of register file 8 to zero and store the result back in register 8.

| Instr Code | Mask (LSH) | Oprd Addr | Oprd Sel EB1- | Mask (MSH) | Destination Selects | | | | | | | | Cn | CF2- CF0 | SIO3- SIO0 | IESIO3- IESIO0 |
|---------------|---------------|--------------|------------------|---------------|---------------------|-------------|-------------------|---------------|-----|------|---------------|-----|----|-------------|---------------|-------------------|
| | | | | | SELMO | WE3- WE0 | SELRF1- SELRF0 | OEY3- OEY0 | OEB | OES | OEY3- OEY0 | OES | | | | |
| 0001 1000 | 1111 | 00 1000 | X 00 | 0000 | 0 | 0000 | 10 | X | X | XXXX | 0 | 0 | X | 110 | 1001 | 0000 |

Assume register file 8 holds A083BEBE (Hex).

Source 0000 1111 0000 1111 0000 1111 0000 1111

Rn ← C3-C0::A3-A0

Source 1010 0000 1000 0011 1011 1110 1011 1110

Sn ← RF(3)n

ALU 1010 0000 1000 0000 1011 0000 1011 1110

Fn ← Sn AND Rn

Destination 1010 0000 1000 0000 1011 0000 1011 1110

RF(8)n ← Fn or Sn[†]

[†]F = ALU result

n = nth byte

Register file 8 gets F if byte selected, S if byte not selected.

FUNCTION

Sets bits in selected bytes of S-bus data using mask in C3-C0::A3-A0.

DESCRIPTION

The register addressed by B5-B0 is both the source and destination for this instruction. The source word is passed on the S bus to the ALU, where it is compared to an 8-bit mask, consisting of a concatenation of the C3-C0 and A3-A0 address ports (C3-C0::A3-A0). The mask is input via the R bus. All bits in the source word that are in the same bit position as ones in the mask are forced to a logical one. Bytes with their $\overline{\text{SIO}}$ inputs programmed low perform the Set Bit instruction. Bytes with their $\overline{\text{SIO}}$ inputs programmed high or floating pass S unaltered.

Available R Bus Source Operands

| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
|---------------|----------------|---------|------------------------------|
| No | No | No | Yes |

Available S Bus Source Operands (MSH)

| RF (B5-B0) | DB-Port | MQ Register |
|---------------|---------|----------------|
| Yes | Yes | Yes |

Available Destination Operands

| RF (C5-C0) | RF (B5-B0) | Y-Port |
|---------------|---------------|--------|
| No | Yes | Yes |

Shift Operations

| ALU | MQ |
|------|------|
| None | None |

Control/Data Signals

| Signal | User Programmable | Use |
|--------------------------|-------------------|-------------|
| SSF | No | Inactive |
| $\overline{\text{SIO0}}$ | Yes | Byte-select |
| $\overline{\text{SIO1}}$ | No | Byte-select |
| $\overline{\text{SIO2}}$ | No | Byte-select |
| $\overline{\text{SIO3}}$ | No | Byte-select |
| Cn | No | Inactive |

Status Signals

| | |
|------|------------------------------------|
| ZERO | = 1 if result (selected bytes) = 0 |
| N | = 0 |
| OVR | = 0 |
| C | = 0 |

EXAMPLE (assumes a 32-bit configuration)

Set bits 3-0 of byte 1 of register file 1 to zero and store the result back in register 1.

| Instr Code I7-I0 | Mask (LSH) A3-A0 | Oprd Addr B5-B0 | Oprd Sel EB1- EA EBO | Mask (MSH) C3-C0 | Destination Selects | | | | | | | | Cn | CF2- CF0 | SIO3- SIO0 | IESIO3- IESIO0 |
|------------------------|------------------------|-----------------------|----------------------------|------------------------|---------------------|----------------|--------|-----|-----|--------------|-----|---|-----|-------------|---------------|-------------------|
| | | | | | WE3- SELMQ | SELRF1- WE0 | SELRF0 | OEA | OEB | OY3- OEY0 | OES | | | | | |
| 0000 1000 | 1111 | 00 0001 | X 00 | 0000 | 0 | 0000 | 10 | X | X | XXXX | 0 | X | 110 | 1101 | 0000 | |

Assume register file 8 holds A083BEBE (Hex).

Source 0000 1111 0000 1111 0000 1111 0000 1111 Rn ← C3-C0::A3-A0

Source 1010 0000 1000 0011 1011 1110 1011 1110 Sn ← RF(1)n

ALU 1010 0000 1000 0011 1011 1111 1011 1110 Fn ← Sn OR Rn

Destination 1010 0000 1000 0011 1011 1111 1011 1110 RF(1)n ← Fn or Sn[†]

[†]F = ALU result
n = nth byte
Register file 1 gets F if byte selected, S if byte not selected.

FUNCTION

Performs arithmetic left shift on result of ALU operation specified in lower nibble of instruction field.

DESCRIPTION

The result of the ALU operation specified in instruction bits I3-I0 is shifted one bit to the left. A zero is filled into bit 0 of the least significant byte of each word unless the $\overline{\text{SIO}}$ input is programmed low; this will force bit 0 to one. Bit 7 is dropped from the most significant byte in each word, which may be 1, 2, or 4 bytes long, depending on the configuration selected.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to the MQ register. If SSF is low, the MQ register will not be altered.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

| ALU Shifter | MQ Shifter |
|-----------------|------------|
| Arithmetic Left | None |

Available Destination Operands (ALU Shifter)

| RF (C5-C0) | RF (B5-B0) | Y-Port |
|---------------|---------------|--------|
| Yes | No | Yes |

Control/Data Signals

| Signal | User Programmable | Use |
|--------------------------|-------------------|--|
| SSF | Yes | Passes shift result if high; passes ALU result if low. Fills a zero in LSB of each word if high; fills a one in LSB if low. |
| $\overline{\text{SIO0}}$ | Yes | |
| $\overline{\text{SIO1}}$ | Yes | |
| $\overline{\text{SIO2}}$ | Yes | |
| $\overline{\text{SIO3}}$ | Yes | |
| Cn | No | Affects arithmetic operation programmed in bits I3-I0 of instruction field. |

Status Signals[†]

ZERO = 1 if result = 0

N = 1 if MSB of result = 1

= 0 if MSB of result = 0

OVR = 1 if signed arithmetic overflow or if MSB XOR MSB-1 = 1 before shift

C = 1 if carry-out condition

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Perform the computation $A = 2(A + B)$, where A and B are single-precision, two's complement numbers. Let A be stored in register 1 and B be input via the DB bus.

| Instr Code 17-10 | Oprd Addr A5-A0 | Oprd Addr B5-B0 | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | Cn | CF0 | SIO3- SIO0 | IESIO3- IESIO0 | SS |
|------------------------|-----------------------|-----------------------|----------------------------|-----------------------|---------------------|-------------|-----------------|-----|-----|---------------|-----|----|-----|---------------|-------------------|----|
| | | | | | SELMO | WE3- WE0 | SELR1- SELR0 | OEA | OEB | OEY3- OEY0 | OES | | | | | |
| 0100 0001 | 00 0001 | XX XXXX | 0 10 | 00 0001 | 0 | 0000 | 10 | X | X | XXXX | 0 | 0 | 110 | 1110 | 0000 | 1 |

Assume register file 1 holds 1308C618 (Hex), DB bus holds 44007530 (Hex).

Source 0001 0011 0000 1000 1100 0110 0001 1000

R ← RF(1)

Source 0100 0100 0000 0000 0111 0101 0011 0000

S ← DB bus

Intermediate
Result 0101 0111 0000 1001 0011 1011 0100 1000

ALU Shifter ← R + S + Cn

Destination 1010 1110 0001 0010 0111 0110 1001 0001

RF(1) ← ALU shift result

| Signal | Programmable | Use |
|--------|--------------|--|
| SIO | Yes | Passes shift result if right; passes ALU result if low. |
| SIO | Yes | Passes zero in LSB of each word if right; passes a |
| SIO | Yes | one in LSB if low. |
| SIO | Yes | |
| SIO | Yes | |
| Cn | No | Affects arithmetic operation specified in bits 13-10 of instruction field. |

FUNCTION

Performs arithmetic left shift on MQ register (LSH) and result of ALU operation (MSH) specified in lower nibble of instruction field.

DESCRIPTION

The result of the ALU operation specified in instruction bits I3-I0 is used as the upper half of a double-precision word, the contents of the MQ register as the lower half.

The contents of the MQ register are shifted one bit to the left. A zero is filled into bit 0 of the least significant byte of each word unless the \overline{SIO} input for the word is set to zero; this will force bit 0 to one. Bit 7 of the most significant byte in the MQ shifter is passed to bit 0 of the least significant byte of the ALU shifter. Bit 7 of the most significant byte in the ALU shifter is dropped.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to the Y MUX and MQ register. If SSF is low, the ALU output and MQ register will not be altered.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

| ALU Shifter | MQ Shifter |
|-----------------|-----------------|
| Arithmetic Left | Arithmetic Left |

Available Destination Operands (ALU Shifter)

| RF (C5-C0) | RF (B5-B0) | Y-Port |
|---------------|---------------|--------|
| Yes | No | Yes |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|-------------------|--|
| SSF | Yes | Passes shift result if high; passes ALU result if low. |
| $\overline{SIO0}$ | Yes | Fills a zero in LSB of each word if high; fills a one in LSB if low. |
| $\overline{SIO1}$ | Yes | |
| $\overline{SIO2}$ | Yes | |
| $\overline{SIO3}$ | Yes | |
| Cn | No | Affects arithmetic operation specified in bits I3-I0 of instruction field. |

Status Signals[†]

ZERO = 1 if result = 0

N = 1 if MSB of result = 1

= 0 if MSB of result = 0

OVR = 1 if signed arithmetic overflow or if MSB XOR MSB-1 = 1 before shift

C = 1 if carry-out condition

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Perform the computation $A = 2(A + B)$, where A and B are two's complement numbers. Let A be a double precision number residing in register 1 (MSH) and the MQ register (LSH). Let B be a single precision number which is input through the DB bus.

| Instr Code | Oprd Addr | Oprd Addr | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2- CF0 | SIO3- SIO0 | IESIO3- IESIO0 | SS |
|---------------|--------------|--------------|----------------------------|-----------------------|---------------------|-------------|-------------------|-----|-----|-------------|-----|---|-----|-------------|---------------|-------------------|----|
| | | | | | SELMQ | WE3- WE0 | SELRF1- SELRFO | OEA | OEB | OY3- OY0 | OES | | | | | | |
| 0101 0001 | 00 0001 | XX XXXX | 0 10 | 00 0001 | 0 | 0000 | 10 | X | X | XXXX | 0 | 0 | 110 | 1110 | 0000 | 1 | |

Assume register file 1 holds 2408C618 (Hex), DB bus holds 26007530 (Hex), and MQ register holds 50A99A0E (Hex).

MSHSource 0010 0100 0000 1000 1100 0110 0001 1000 $R \leftarrow RF(1)$ Source 0010 0110 0000 0000 0111 0101 0011 0000 $S \leftarrow DB \text{ bus}$ Intermediate Result 0100 1010 0000 1001 0011 1011 0100 1000 $ALU \text{ Shifter} \leftarrow R + S + Cn$ Destination 1001 0100 0001 0010 0111 0110 1001 0000 $RF(1) \leftarrow ALU \text{ shift register}$ **LSH**Source 0101 0000 1010 1001 1001 1010 0000 1110 $MQ \text{ shifter} \leftarrow MQ \text{ register}$ Destination 1010 0001 0101 0011 0011 0100 0001 1101 $MQ \text{ register} \leftarrow MQ \text{ shift result}$

FUNCTION

Performs circular left shift on result of ALU operation specified in lower nibble of instruction field.

DESCRIPTION

The result of the ALU operation specified in instruction bits I3-I0 is rotated one bit to the left. Bit 7 of the most significant byte in each word is passed to bit 0 of the least significant byte in the word, which may be 1, 2, or 4 bytes long.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to Y MUX. If SSF is low, F is passed unaltered.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

| ALU Shifter | MQ Shifter |
|---------------|------------|
| Circular Left | None |

Available Destination Operands (ALU Shifter)

| RF (C5-C0) | RF (B5-B0) | Y-Port |
|---------------|---------------|--------|
| Yes | No | Yes |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|-------------------|--|
| SSF | Yes | Passes shift result if high; passes ALU result if low. |
| $\overline{SI00}$ | No | Bit 7 of ALU result |
| $\overline{SI01}$ | No | Bit 15 of ALU result |
| $\overline{SI02}$ | No | Bit 23 of ALU result |
| $\overline{SI03}$ | No | Bit 31 of ALU result |
| Cn | No | Affects arithmetic operation specified in bits I3-I0 of instruction field. |

Status Signals[†]

ZERO = 1 if result = 0
 N = 1 if MSB of result = 1
 = 0 if MSB of result = 0
 OVR = 1 if signed arithmetic overflow
 C = 1 if carry-out condition

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Perform a circular left shift of register 6 and store the result in register 1.

| Instr Code 17-10 | Oprd Addr A5-A0 | Oprd Addr B5-B0 | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2- CF0 | SSF |
|------------------------|-----------------------|-----------------------|----------------------------|-----------------------|---------------------|----------------|--------|-----|-----|--------------|-----|---|-----|-------------|-----|
| | | | | | WE3- SELMO | SELRF1- WEO | SELRF0 | OEA | OEB | OEY0 OEY0 | OES | | | | |
| 0110 0110 | 00 0110 | XX XXXX | 0 00 | 00 0001 | 0 | 0000 | 10 | X | X | XXXX | 0 | 0 | 110 | 1 | |

Assume register file 6 holds 3788C618 (Hex).

Source 0011 0111 1000 1000 1100 0110 0001 1000

R ← RF(6)

Intermediate Result 0011 0111 1000 1000 1100 0110 0001 1000

ALU Shifter ← R + Cn

Destination 0110 1111 0001 0001 1000 1100 0011 0000

RF(1) ← ALU shifter result

| Signal | Programmable | Use |
|--------|--------------|--|
| SSF | Yes | Passes shift result if right; passes ALU result if low. |
| St00 | No | Bit 7 of ALU result |
| St01 | No | Bit 15 of ALU result |
| St02 | No | Bit 23 of ALU result |
| St03 | No | Bit 31 of ALU result |
| Cn | No | Affects arithmetic operation specified in bits 13-10 of instruction field. |

FUNCTION

Performs circular left shift on MQ register (LSH) and result of ALU operation specified in lower nibble of instruction field (MSH).

DESCRIPTION

The result of the ALU operation specified in instruction bits I3-I0 is used as the upper half of a double-precision word, the contents of the MQ register as the lower half.

The contents of the MQ and ALU registers are rotated one bit to the left. Bit 7 of the most significant byte in the MQ shifter is passed to bit 0 of the least significant byte of the ALU shifter. Bit 7 of the most significant byte is passed to bit 0 of the least significant byte in the MQ shifter.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to Y MUX. If SSF is low, F is passed unaltered and the MQ register is not changed.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

| ALU Shifter | MQ Shifter |
|---------------|---------------|
| Circular Left | Circular Left |

Available Destination Operands (ALU Shifter)

| RF (C5-C0) | RF (B5-B0) | Y-Port |
|---------------|---------------|--------|
| Yes | No | Yes |

Control/Data Signals

| Signal | User Programmable | Use |
|--------------------------|-------------------|--|
| SSF | Yes | Passes shift result if high; passes ALU result if low. |
| $\overline{\text{SIO0}}$ | No | Bit 7 of ALU result |
| $\overline{\text{SIO1}}$ | No | Bit 15 of ALU result |
| $\overline{\text{SIO2}}$ | No | Bit 23 of ALU result |
| $\overline{\text{SIO3}}$ | No | Bit 31 of ALU result |
| Cn | No | Affects arithmetic operation specified in bits I3-I0 of instruction field. |

Status Signals[†]

ZERO = 1 if result = 0
 N = 1 if MSB of result = 1
 = 0 if MSB of result = 0
 OVR = 1 if signed arithmetic overflow
 C = 1 if carry-out condition

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Perform a circular left double precision shift of data in register 6 (MSH) and MQ (LSH), and store the result back in register 6 and the MQ register.

| Instr Code | Oprd Addr | Oprd Addr | Oprd Sel EB1- EA EB0 | Dest Addr C5-C0 | Destination Selects | | | | | | | Cn | CF2- CF0 | SSF |
|---------------|--------------|--------------|----------------------------|-----------------------|---------------------|----------------|--------|-------------|-------------|-------------|-------------|----|-------------|-----|
| | | | | | WE3- SELMO | SELRF1- WE0 | SELRF0 | OE3- OE0 | OE2- OE1 | OE1- OE0 | OE0- OE0 | | | |
| 0111 0110 | 00 0110 | XX XXXX | 0 00 | 00 0110 | 0 | 0000 | 10 | X | X | XXXX | 0 | 0 | 110 | 1 |

Assume register file 6 holds 3708C618 (Hex) and MQ register holds 50A99A0E (Hex).

MSH

Source 0011 0111 0000 1000 1100 0110 0001 1000 $R \leftarrow RF(6)$

Intermediate Result 0011 0111 0000 1000 1100 0110 0001 1000 $ALU\ Shifter \leftarrow R + Cn$

Destination 0110 1111 0001 0001 1000 1100 0011 0000 $RF(6) \leftarrow ALU\ shifter\ result$

LSH

Source 0101 0000 1010 1001 1001 1010 0000 1110 $MQ\ register \leftarrow MQ\ register$

Destination 1010 0001 0101 0011 0011 0100 0001 1100 $MQ\ register \leftarrow MQ\ shift\ result$

FUNCTION

Converts data on the S bus from sign magnitude to two's complement or vice versa.

DESCRIPTION

The S bus provides the source word for this instruction. The number is converted by inverting S and adding the result to the carry-in, which should be programmed high for proper conversion; the sign bit of the result is then inverted. An error condition will occur if the source word is a negative zero (negative sign and zero magnitude). In this case, SMTC generates a positive zero, and the OVR pin is set high to reflect an illegal conversion.

The sign bit of the selected operand in the most significant byte is tested; if it is high, the converted number is passed to the destination. Otherwise the operand is passed unaltered.

Available R Bus Source Operands

| | | | |
|---------------|----------------|---------|------------------------------|
| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
| No | No | No | No |

Available S Bus Source Operands

| | | |
|---------------|---------|----------------|
| RF (B5-B0) | DB-Port | MQ Register |
| Yes | Yes | Yes |

Available Destination Operands Shift Operations

| | | |
|---------------|---------------|--------|
| RF (C5-C0) | RF (B5-B0) | Y-Port |
| Yes | No | Yes |

| | |
|------|------|
| ALU | MQ |
| None | None |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|-------------------|---|
| SSF | No | Inactive |
| $\overline{SIO0}$ | No | Inactive |
| $\overline{SIO1}$ | No | Inactive |
| $\overline{SIO2}$ | No | Inactive |
| $\overline{SIO3}$ | No | Inactive |
| Cn | Yes | Should be programmed high for proper conversion |

Status Signals

ZERO = 1 if result = 0

N = 1 if MSB = 1

OVR = 1 if input of most significant byte is 80 (Hex) and results in all other bytes are 00 (Hex).

C = 1 if S = 0

EXAMPLES (assumes a 32-bit configuration)

Convert the two's complement number in register 1 to sign magnitude representation and store the result in register 4.

| Instr Code | Oprd Addr | Oprd Addr | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2- CFO |
|------------|-----------|-----------|-------------------------|-----------------|---------------------|-------------------|--------------|-------------|---------------|-----|--|--|----|-------------|
| | | | | | WE3- WEO | SELRF1- SELRFO | OE3- OEAO | OE2- OEB | OEY3- OEYO | OES | | | | |
| 0101 1000 | XX XXXX | 00 0001 | X 00 | 00 0100 | 0 0000 | 10 | X | X | XXXX | 0 | | | 1 | 110 |

Example 1: Assume register file 1 holds C3F6D840 (Hex).

Source 1100 0011 1111 0110 1101 1000 0100 0000 $S \leftarrow RF(1)$

Destination 1011 1100 0000 1001 0010 0111 1100 0000 $RF(4) \leftarrow S' + Cn$

Example 2: Assume register file 1 holds 550927C0 (Hex).

Source 0101 0101 0000 1001 0010 0111 1100 0000 $S \leftarrow RF(1)$

Destination 0101 0101 0000 1001 0010 0111 1100 0000 $RF(4) \leftarrow S$

FUNCTION

Computes one of N-1 signed or N mixed multiplication iterations for computing an N-bit by N-bit product. Algorithms for signed and mixed multiplication using this instruction are given in the "Other Arithmetic Instructions" section.

DESCRIPTION

SMULI checks to determine whether the multiplicand should be added with the present partial product. The instruction evaluates:

$F \leftarrow R + S + C_n$ if the addition is required
 $F \leftarrow S$ if no addition is required

A double precision right shift is performed. Bit 0 of the least significant byte of the ALU shifter is passed to bit 7 of the most significant byte of the MQ shifter; carry-out is passed to the most significant bit of the ALU shifter.

The S bus should be loaded with the contents of an accumulator and the R bus with the multiplicand. The Y bus result should be written back to the accumulator after each iteration of UMULI. The accumulator should be cleared and the MQ register loaded with the multiplier before the first iteration.

Available R Bus Source Operands

| | | | |
|---------------|----------------|---------|------------------------------|
| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
| Yes | No | Yes | No |

Recommended S Bus Source Operands

| | | |
|---------------|---------|----------------|
| RF (B5-B0) | DB-Port | MQ Register |
| Yes | Yes | No |

Recommended Destination Operands Shift Operations

| | | |
|---------------|---------------|--------|
| RF (C5-C0) | RF (B5-B0) | Y-Port |
| Yes | No | No |

| | |
|-------|-------|
| ALU | MQ |
| Right | Right |

Control/Data Signals

| Signal | User Programmable | Use |
|--------------------------|-------------------|---|
| SSF | No | Inactive |
| $\overline{\text{STO0}}$ | No | Passes LSB from ALU shifter to MSB of MQ shifter. |
| $\overline{\text{STO1}}$ | No | |
| $\overline{\text{STO2}}$ | No | |
| $\overline{\text{STO3}}$ | No | |
| Cn | Yes | Should be programmed low |

Status Signals

| | |
|------|-------------------|
| ZERO | = 1 if result = 0 |
| N | = 1 if MSB = 1 |
| OVR | = 0 |
| C | = 1 if carry-out |

FUNCTION

Performs the final iteration for computing an N-bit by N-bit signed product. An algorithm for signed multiplication using this instruction is given in the "Other Arithmetic Instructions" section.

DESCRIPTION

SMULT checks the present multiplier bit (the least significant bit of the MQ register) to determine whether the multiplicand should be added with the present partial product. The instruction evaluates:

$$F \leftarrow R' + S + C_n \quad \text{if the addition is required}$$

$$F \leftarrow S \quad \text{if no addition is required}$$

with the correct sign in the product.

A double precision right shift is performed. Bit 0 of the least significant byte of the ALU shifter is passed to bit 7 of the most significant byte of the MQ shifter.

The S bus should be loaded with the contents of an register file holding the previous iteration result; the R bus must be loaded with the multiplicand. After executing SMULT, the Y bus contains the most significant half of the product, and MQ contains the least significant half.

Available R Bus Source Operands

| | | | |
|---------------|----------------|---------|------------------------------|
| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
| Yes | No | Yes | No |

Recommended S Bus Source Operands

| | | |
|---------------|---------|----------------|
| RF (B5-B0) | DB-Port | MQ Register |
| Yes | Yes | No |

Available Destination Operands Shift Operations

| | | |
|---------------|---------------|--------|
| RF (C5-C0) | RF (B5-B0) | Y-Port |
| Yes | No | No |

| | |
|-------|-------|
| ALU | MQ |
| Right | Right |

Control/Data Signals

| Signal | User Programmable | Use |
|--------------------------|-------------------|---|
| SSF | No | Inactive |
| $\overline{\text{SIO0}}$ | No | Passes LSB from ALU shifter to MSB of MQ shifter. |
| $\overline{\text{SIO1}}$ | No | |
| $\overline{\text{SIO2}}$ | No | |
| $\overline{\text{SIO3}}$ | No | |
| Cn | Yes | Should be programmed high |

Status Signals

| | |
|------|-------------------|
| ZERO | = 1 if result = 0 |
| N | = 1 if MSB = 1 |
| OVR | = 0 |
| C | = 1 if carry-out |

FUNCTION

Tests the two most significant bits of the MQ register. If they are the same, shifts the number to the left.

DESCRIPTION

This instruction is used to normalize a two's complement number in the MQ register by shifting the number one bit position to the left and filling a zero into the LSB (unless the \overline{SIO} input for that word is low). Data on the S bus is added to the carry, permitting the number of shifts performed to be counted and stored in one of the register files.

The shift and the S bus increment are inhibited whenever normalization is attempted on a number already normalized. Normalization is complete when overflow occurs.

Available R Bus Source Operands

| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
|---------------|----------------|---------|------------------------------|
| No | No | No | No |

Available S Bus Source Operands (Count)

| RF (B5-B0) | DB-Port | MQ Register |
|---------------|---------|----------------|
| Yes | No | No |

| | |
|---|---|
| Available Destination Operands (Count) | Shift Operations (Conditional) |
|---|---|

| RF (C5-C0) | RF (B5-B0) | Y-Port |
|---------------|---------------|--------|
| Yes | No | Yes |

| ALU | MQ |
|-----|------|
| No | Left |

Control/Data Signals

| User | | |
|--------------------------|--------------|---|
| Signal | Programmable | Use |
| SSF | No | Inactive |
| $\overline{\text{SIO0}}$ | No | Passes internally generated end-fill bit. |
| $\overline{\text{SIO1}}$ | No | |
| $\overline{\text{SIO2}}$ | No | |
| $\overline{\text{SIO3}}$ | No | |
| Cn | Yes | Increments S bus (shift count) if set to one. |

Status Signals

ZERO = 1 if result = 0
 N = 1 if MSB of MQ register = 1
 OVR = 1 if MSB of MQ register XOR 2nd MSB = 1
 C = 1 if carry-out = 1

EXAMPLE (assumes a 32-bit configuration)

Normalize the number in the MQ register, storing the number of shifts in register 3.

| Instr Code | Oprd Addr | Oprd Addr | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF0 |
|---------------|--------------|--------------|----------------------------|-----------------------|---------------------|-------------|-------------------|-------------|-------------|--------------|-------------|---|-----|-----|
| | | | | | SELMO | WE3- WEO | SELRF1- SELRFO | OE3- OEA | OE2- OEB | OE1- OEYO | OE0- OES | | | |
| 0010 0000 | XX XXXX | 00 0011 | X 00 | 00 0011 | 0 | 0000 | 10 | X | X | XXXX | 0 | 1 | 110 | |

Assume register file 3 holds 00000003 (Hex) and MQ register holds 3699D84E (Hex).

Operand

Source 0011 0110 1001 1001 1101 1000 0100 1110 MQ shifter \leftarrow MQ register

Destination 0110 1101 0011 0011 1011 0000 1001 1100 MQ register \leftarrow MQ shifter

Count

Source 0000 0000 0000 0000 0000 0000 0000 0011 S \leftarrow RF(3)

Destination 0000 0000 0000 0000 0000 0000 0000 0100 RF(3) \leftarrow S + Cn

FUNCTION

Performs arithmetic right shift on result of ALU operation specified in lower nibble of instruction field.

DESCRIPTION

The result of the ALU operation specified in instruction bits I3-I0 is shifted one bit to the right. The sign bit of the most significant byte is retained unless it is inverted as a result of overflow. Bit 0 of the least significant byte is dropped.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to the Y MUX. If SSF is low, the ALU result will be passed unshifted to the Y MUX.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

| ALU Shifter | MQ Shifter |
|------------------|------------|
| Arithmetic Right | None |

Available Destination Operands (ALU Shifter)

| RF (C5-C0) | RF (B5-B0) | Y-Port |
|---------------|---------------|--------|
| Yes | No | Yes |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|-------------------|---|
| SSF | Yes | Passes shifted output if high; passes ALU result if low. |
| $\overline{SIO0}$ | No | LSB is shifted out from each word, which may be 1, 2, or 4 bytes long depending on selected configuration |
| $\overline{SIO1}$ | No | |
| $\overline{SIO2}$ | No | |
| $\overline{SIO3}$ | No | |
| Cn | No | Affects arithmetic operation specified in bits I3-I0 of instruction field. |

Status Signals†

ZERO = 1 if result = 0

N = 1 if MSB of result = 1

= 0 if MSB of result = 0

OVR = 0

C = 1 if carry-out condition

†C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Perform the computation $A = (A + B)/2$, where A and B are single-precision numbers. Let A reside in register 1 and B be input via the DB bus.

| Instr Code I7-I0 | Oprd Addr A5-A0 | Oprd Addr B5-B0 | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2- CF0 | SSF |
|------------------------|-----------------------|-----------------------|----------------------------|-----------------------|---------------------|------|--------|-----|-----|------|------|------|----|-------------|-----|
| | | | | | SELMO | WE0 | SELRF0 | OE0 | OE1 | OEY0 | OEY1 | OEY2 | | | |
| 0000 0001 | 00 0001 | XX XXXX | 0 10 | 00 0001 | 0 | 0000 | 10 | X | X | XXXX | 0 | 0 | 0 | 110 | 1 |

Assume register file 1 holds 6A08C618 (Hex) and DB bus holds 51007530 (Hex).

Source

0110 1010 0000 1000 1100 0110 0001 1000

R ← RF(1)

Source

0101 0001 0000 0000 0111 0101 0011 0000

S ← DB bus

Intermediate†

Result

1011 1011 0000 1001 0011 1011 0100 1000

ALU Shifter ← R + S + Cn

Destination

0101 1101 1000 0100 1001 1101 1010 0100

RF(1) ← ALU shift result

†After the intermediate operation (ADD), overflow has occurred and OVR status signal is set high. When the arithmetic right shift is executed, the sign bit is corrected (see Table 16 for shift definition notes).

FUNCTION

Performs arithmetic right shift on MQ register (LSH) and result of ALU operation (MSH) specified in lower nibble of instruction field.

DESCRIPTION

The result of the ALU operation specified in instruction bits I3-I0 is used as the upper half of a double precision word, the contents of the MQ register as the lower half.

The contents of the ALU are shifted one bit to the right. The sign bit of the most significant byte is retained unless the sign bit is inverted as a result of overflow. Bit 0 of the least significant byte in the ALU shifter is passed to bit 7 of the most significant byte of the MQ register. Bit 0 of the MQ register's least significant byte is dropped.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to the Y MUX. If SSF is low, the ALU result will be passed unshifted to the Y MUX.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

| ALU Shifter | MQ Shifter |
|------------------|------------------|
| Arithmetic Right | Arithmetic Right |

Available Destination Operands (ALU Shifter)

| RF (C5-C0) | RF (B5-B0) | Y-Port |
|---------------|---------------|--------|
| Yes | No | Yes |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|-------------------|--|
| SSF | Yes | Passes shifted output if high; passes ALU result if low. |
| $\overline{SIO0}$ | No | LSB of ALU shifter is passed to MSB of MQ shifter, and LSB of MQ shifter is dropped. |
| $\overline{SIO1}$ | No | |
| $\overline{SIO2}$ | No | |
| $\overline{SIO3}$ | No | |
| Cn | No | Affects arithmetic operation specified in bits I3-I0 of instruction field. |

Status Signals[†]

ZERO = 1 if result = 0
 N = 1 if MSB of result = 1
 = 0 if MSB of result = 0
 OVR = 0
 C = 1 if carry-out condition

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Perform the computation $A = (A + B)/2$, where A and B are two's complement numbers. Let A be a double precision number residing in register 1 (MSH) and MQ (LSH). Let B be a single precision number which is input through the DB bus.

| Instr Code | Oprd Addr | Oprd Addr | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2- CF0 | SSF |
|---------------|--------------|--------------|----------------------------|-----------------------|---------------------|----------------|-----------------|---------------|--------------|--------------|---|---|-----|-------------|-----|
| | | | | | WE3- SELMO | SELRF1- WE0 | OEY3- SELRFO | OEY0- OEY0 | OEY3- OES | OEY0- OES | | | | | |
| 0001 0001 | 00 0001 | XX XXXX | 0 10 | 00 0001 | 0 | 0000 | 10 | X | X | XXXX | 0 | 0 | 110 | 1 | |

Assume register file 1 holds 4A08C618 (Hex), and DB bus holds 51007530 (Hex), and MQ register holds 17299A0F (Hex).

MSH

Source 0100 1010 0000 1000 1100 0110 0001 1000 R ← RF(1)

Source 0101 0001 0000 0000 0111 0101 0011 0000 S ← DB bus

Intermediate[‡] Result 1001 1011 0000 1001 0011 1011 0100 1000 ALU Shifter ← R + S + Cn

Destination 0100 1101 1000 0100 1001 1101 1010 0100 RF(1) ← ALU shift result

LSH

Source 0001 0111 0010 1001 1001 1010 0000 1111 MQ shifter ← MQ register

Destination 0000 1011 1001 0100 1100 1101 0000 0111 MQ register ← MQ shift result

[‡]After the intermediate operation (ADD), overflow has occurred and OVR status signal is set high. When the arithmetic right shift is executed, the sign bit is corrected (see Table 16 for shift definition notes).

FUNCTION

Performs circular right shift on result of ALU operation specified in lower nibble of instruction field.

DESCRIPTION

The result of the ALU operation specified in instruction bits I3-I0 is shifted one bit to the right. Bit 0 of the least significant byte is passed to bit 7 of the most significant byte in the same word, which may be 1, 2, or 4 bytes long depending on the selected configuration.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to the Y MUX. If SSF is low, the ALU result will be passed unshifted to the Y MUX.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

| ALU Shifter | MQ Shifter |
|----------------|------------|
| Circular Right | None |

Available Destination Operands (ALU Shifter)

| RF (C5-C0) | RF (B5-B0) | Y-Port |
|---------------|---------------|--------|
| Yes | No | Yes |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|-------------------|--|
| SSF | Yes | Passes shift result if high; passes ALU result if low. |
| $\overline{SI00}$ | No | Rotates LSB to MSB of the same word, which may be 1, 2, or 4 bytes long depending on configuration |
| $\overline{SI01}$ | No | |
| $\overline{SI02}$ | No | |
| $\overline{SI03}$ | No | |
| Cn | No | Affects arithmetic operation specified in bits I3-I0 of instruction field. |

Status Signals[†]

| | |
|------|--|
| ZERO | = 1 if result = 0 |
| N | = 1 if MSB of result = 1 = 0 if MSB of result = 0 |
| OVR | = 1 if signed arithmetic overflow |
| C | = 1 if carry-out condition |

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Perform a circular right shift of register 6 and store the result in register 1.

| Instr | Oprd | Oprd | Oprd Sel | Dest | Destination Selects | | | | | | | | | | |
|-----------|---------|---------|----------------|---------|---------------------|----------------|--------|-------------|-------------|--------------|-------------|------|-------------|-----|--|
| Code | Addr | Addr | EB1- EA EBO | Addr | WE3- SELMQ | SELRF1- WE0 | SELRF0 | OE3- OEA | OE2- OEB | OE1- OEY0 | OE0- OES | Cn | CF2- CF0 | SSF | |
| 17-10 | A5-A0 | B5-B0 | EA EBO | C5-C0 | SEL | MQ | WE0 | SEL | RF0 | OEA | OEB | OEY0 | OES | | |
| 1000 0110 | 00 0110 | XX XXXX | 0 XX | 00 0001 | 0 | 0000 | 10 | X | X | XXXX | 0 | 0 | 110 | 1 | |

Assume register file 6 holds 3788C618 (Hex).

| | | |
|---------------------|---|--------------------------|
| Source | 0011 0111 1000 1000 1100 0110 0001 1000 | R ← RF(6) |
| Intermediate Result | 0011 0111 1000 1000 1100 0110 0001 1000 | ALU Shifter ← R + Cn |
| Destination | 0001 1011 1100 0100 0110 0011 0000 1100 | RF(1) ← ALU shift result |

| Signal | User Programmable | Use |
|--------|-------------------|---|
| SZP | Yes | Passes shift result if right; passes ALU result and retains MG register if low. |
| SI00 | No | Rotates LSB of ALU shifter to MSB of MG shifter. |
| SI01 | No | Rotates LSB of MG shifter to MSB of ALU shifter. |
| SI02 | No | |
| SI03 | No | |
| Cn | No | Affects arithmetic operation specified in bits 13-10 of instruction field. |

FUNCTION

Performs circular right shift on MQ register (LSH) and result of ALU operation (MSH) specified in lower nibble of instruction field.

DESCRIPTION

The result of the ALU operation specified in instruction bits I3-I0 is used as the upper half of a double precision word, the contents of the MQ register as the lower half.

The contents of the ALU and MQ shifters are rotated one bit to the right. Bit 0 of the least significant byte in the ALU shifter is passed to bit 7 of the most significant byte of the MQ shifter. Bit 0 of the least significant byte is passed to bit 7 of the most significant byte of the ALU shifter.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to the Y MXU and MQ register. If SSF is low, the Y MUX and MQ register will not be altered.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

| ALU Shifter | MQ Shifter |
|----------------|----------------|
| Circular Right | Circular Right |

Available Destination Operands (ALU Shifter)

| RF (C5-C0) | RF (B5-B0) | Y-Port |
|---------------|---------------|--------|
| Yes | No | Yes |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|-------------------|--|
| SSF | Yes | Passes shift result if high; passes ALU result and retains MQ register if low. |
| $\overline{SIO0}$ | No | Rotates LSB of ALU shifter to MSB of MQ shifter, and LSB of MQ shifter to MSB of ALU shifter |
| $\overline{SIO1}$ | No | |
| $\overline{SIO2}$ | No | |
| $\overline{SIO3}$ | No | |
| Cn | No | Affects arithmetic operation specified in bits I3-I0 of instruction field. |

Status Signals[†]

ZERO = 1 if result = 0
 N = 1 if MSB of result = 1
 = 0 if MSB of result = 0
 OVR = 1 if signed arithmetic overflow
 C = 1 if carry-out condition

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Perform a circular right double precision shift of the data in register 6 (MSH) and MQ (LSH), and store the result back in register 6 and the MQ register.

| Instr Code | Oprd Addr | Oprd Addr | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2- CF0 |
|---------------|--------------|--------------|----------------------------|-----------------------|---------------------|----------------|--------|-------------|-------------|-------------|-------------|-------------|----|-------------|
| | | | | | WE3- SELMQ | SELRF1- WE0 | SELRF0 | OE3- OE0 | OE2- OE1 | OE1- OE0 | OE0- OE0 | OE0- OE0 | | |
| 1001 0110 | 00 0110 | XX XXXX | 0 XX | 00 0110 | 0 | 0000 | 10 | X | X | XXXX | 0 | 0 | 0 | 110 |

Assume register file 6 holds 3788C618 (Hex) and MQ register holds 50A99A0F (Hex).

MSH

Source 0011 0111 0000 1000 1100 0110 0001 1000 R ← RF(6)

Intermediate Result 0011 0111 0000 1000 1100 0110 0001 1000 ALU shifter ← R + Cn

Destination 1001 1011 1000 0100 0110 0011 0000 1100 RF(6) ← ALU shift result

LSH

Source 0101 0000 1010 1001 1001 1010 0000 1111 MQ shifter ← MQ register

Destination 0010 1000 0101 0100 1100 1101 0000 0111 MQ register ← MQ shift result

FUNCTION

Performs logical right shift on result of ALU operation specified in lower nibble of instruction field.

DESCRIPTION

The result of the ALU operation specified in instruction bits I3-I0 is shifted one bit to the right. A zero is placed in the bit 7 of the most significant byte of each word unless the $\overline{\text{SIO}}$ input for the word is programmed low; this will force the sign bit to one. The LSB is dropped from the word, which may be 1, 2, or 4 bytes long depending on selected configuration.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to the Y MUX. If SSF is low, the ALU result will be passed unshifted to the Y MUX.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

| ALU Shifter | MQ Shifter |
|---------------|------------|
| Logical Right | None |

Available Destination Operands (ALU Shifter)

| RF (C5-C0) | RF (B5-B0) | Y-Port |
|---------------|---------------|--------|
| Yes | No | Yes |

Control/Data Signals[‡]

| Signal | User Programmable | Use |
|--------------------------|-------------------|---|
| SSF | | Passes shift result if high or floating; passes ALU result if low. |
| $\overline{\text{SIO0}}$ | Yes | Fills a zero in MSB of the word if high or floating; fills a one in MSB if low. |
| $\overline{\text{SIO1}}$ | Yes | |
| $\overline{\text{SIO2}}$ | Yes | |
| $\overline{\text{SIO3}}$ | Yes | |
| Cn | | Inactive |

[‡]Cn is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Perform a logical right single precision shift on data on the DA bus, and store the result in register 1.

| Instr Code | Oprd Addr | Oprd Addr | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF0 | SIO3- SIO0 | IESIO3- IESIO0 | SSF |
|---------------|--------------|--------------|----------------------------|-----------------------|---------------------|-------------|--------|------|-----|---------------|-----|---|----|-----|---------------|-------------------|-----|
| | | | | | SELMO | WE3- WEO | SELRFO | OEAE | OEB | OEY3- OEYO | OES | | | | | | |
| 17-10 | A5-A0 | B5-B0 | EA EBO | C5-C0 | SELMO | WE0 | SELRFO | OEAE | OEB | OEYO | OES | 0 | 0 | 110 | XXX1 | 0000 | 1 |

Assume DA bus holds 2DA8C615.

Source

0010 1101 1010 1000 1100 0110 0001 0101

R ← DA bus

Intermediate Result

0010 1101 1010 1000 1100 0110 0001 0101

ALU Shifter ← R + Cn

Destination

0001 0110 1101 0100 0110 0011 0000 1010

RF(1) ← ALU shift result

FUNCTION

Performs logical right shift on MQ register (LSH) and result of ALU operation (MSH) specified in lower nibble of instruction field.

DESCRIPTION

The result of the ALU operation specified in instruction bits I3-I0 is used as the upper half of a double precision word, the contents of the MQ register as the lower half.

The ALU result is shifted one bit to the right. A zero is placed in the sign bit of the most significant byte unless the \overline{SIO} input for that word is programmed low; this will force the sign bit to one. Bit 0 of the least significant byte is passed to bit 7 of the most significant byte of the MQ shifter. Bit 0 of the least significant byte of the MQ shifter is dropped.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to the Y MUX and MQ register. If SSF is low, the ALU result and MQ register will not be altered.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

| | |
|---------------|---------------|
| ALU Shifter | MQ Shifter |
| Logical Right | Logical Right |

Available Destination Operands (ALU Shifter)

| | | |
|---------------|---------------|--------|
| RF (C5-C0) | RF (B5-B0) | Y-Port |
| Yes | No | Yes |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|-------------------|--|
| SSF | Yes | Passes shift result if high; passes ALU result and retains MQ |
| $\overline{SIO0}$ | Yes | Fills a zero in MSB if high or floating; |
| $\overline{SIO1}$ | Yes | fills a one MSB if low. |
| $\overline{SIO2}$ | Yes | |
| $\overline{SIO3}$ | Yes | |
| Cn | No | Affects arithmetic operation specified in bits I3-I0 of instruction field. |

Status Signals[†]

ZERO = 1 if result = 0
 N = 1 if MSB of result = 1
 = 0 if MSB of result = 0
 OVR = 1 if signed arithmetic overflow
 C = 1 if carry-out condition

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Perform a logical right double precision shift of the data in register 1 (MSH) and MQ (LSH), filling a one into the most significant bit, and store the result back in register 1 and the MQ register.

| Instr Code | Oprd Addr | Oprd Addr | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | SELMO | Destination Selects | | | | | | | | Cn | CF0 | SIO3- SIO0 | IESIO3- IESIO0 |
|---------------|--------------|--------------|----------------------------|-----------------------|-------|---------------------|-------------------|-------------|---------------|------|---|---|-----|------|------|---------------|-------------------|
| | | | | | | WE3- WE0 | SELRF1- SELRF0 | OE3- OE0 | OEY3- OEY0 | OES | | | | | | | |
| 0011 0110 | XX XXXX | 00 0001 | X 00 | 00 0001 | 0 | 0000 | 10 | X | X | XXXX | 0 | 0 | 110 | 1110 | 0000 | | |

Assume register file 1 holds 2DA8C615 (Hex) and MQ register holds 50A99A0E (Hex).

MSH

Source 0010 1101 1010 1000 1100 0110 0001 0101 $R \leftarrow RF(1)$

Intermediate Result 0010 1101 1010 1000 1100 0110 0001 0101 ALU Shifter $\leftarrow S + Cn$

Destination 1001 0110 1101 0100 0110 0011 0000 1010 $RF(1) \leftarrow$ ALU shift result

LSH

Source 0101 0000 1010 1001 1001 1010 0000 1110 MQ shifter \leftarrow MQ register

Destination 1010 1000 0101 0100 1100 1101 0000 0111 MQ register \leftarrow MQ shift result

FUNCTION

Subtracts four-bit immediate data on A3-A0 with carry from S-bus data.

DESCRIPTION

Immediate data in the range 0 to 15, supplied by the user at A3-A0, is inverted and added with carry to S.

Available R Bus Source Operands (Constant)

| | | | |
|---------------|----------------|---------|------------------------------|
| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
| No | Yes | No | No |

Available S Bus Source Operands

| | | |
|---------------|---------|----------------|
| RF (B5-B0) | DB-Port | MQ Register |
| Yes | Yes | Yes |

Available Destination Operands Shift Operations

| | | | | |
|---------------|---------------|--------|------|------|
| RF (C5-C0) | RF (B5-B0) | Y-Port | ALU | MQ |
| Yes | No | Yes | None | None |

Control/Data Signals

| Signal | User Programmable | Use |
|--------------------------|-------------------|--|
| SSF | No | Inactive |
| $\overline{\text{SIO0}}$ | No | Inactive |
| $\overline{\text{SIO1}}$ | No | Inactive |
| $\overline{\text{SIO2}}$ | No | Inactive |
| $\overline{\text{SIO3}}$ | No | Inactive |
| Cn | Yes | Two's complement subtraction if programmed high. |

Status Signals

| | |
|------|-----------------------------------|
| ZERO | = 1 if result = 0 |
| N | = 1 if MSB = 1 |
| OVR | = 1 if arithmetic signed overflow |
| C | = 1 if carry-out |

EXAMPLE (assumes a 32-bit configuration)

Subtract the value 12 from data on the DB bus, and store the result into register file 1.

| Instr Code I7-I0 | Oprd Addr A5-A0 | Oprd Addr B5-B0 | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2- CF0 |
|------------------------|-----------------------|-----------------------|----------------------------|-----------------------|---------------------|----------------|-------------------|-----|-----|--------------|-----|---|-----|-------------|
| | | | | | WE3- SELMO | SELRF1- WEO | SELRF0- SELRFO | OEA | OEB | OY3- OEYO | OES | | | |
| 0111 1000 | 00 1100 | XX XXXX | X 10 | 00 0001 | 0 | 0000 | 10 | X | X | XXXX | 0 | 1 | 110 | |

Assume bits A3-A0 hold C (Hex) and DB bus holds 24000100 (Hex).

| | | |
|-------------|---|---------------------|
| Source | 0000 0000 0000 0000 0000 0000 0000 1100 | R ← A3-A0 |
| Source | 0010 0100 0000 0000 0000 0001 0000 0000 | S ← DB bus |
| Destination | 0010 0100 0000 0000 0000 0000 1111 0100 | RF(1) ← R' + S + Cn |

SUBR**Subtract R with Carry (R' + S + Cn)**

*

2

FUNCTION

Subtracts data on the R bus from S with carry.

DESCRIPTION

Data on the R bus is subtracted with carry from data on the S bus. The result appears at the ALU and MQ shifters.

*The result of this instruction can be shifted in the same microcycle by specifying a shift instruction in the upper nibble (I7-I4) of the instruction field. The result may also be passed without shift. Possible instructions are listed in Table 15.

Available R Bus Source Operands

| | | | |
|---------------|----------------|---------|-----------------------------|
| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 : A3-A0 Mask |
| Yes | No | Yes | No |

Available S Bus Source Operands

| | | |
|---------------|---------|----------------|
| RF (B5-B0) | DB-Port | MQ Register |
| Yes | Yes | Yes |

Available Destination Operands

| | | | | |
|---------------|---------------|--------|----------------|---------------|
| RF (C5-C0) | RF (B5-B0) | Y-Port | ALU Shifter | MQ Shifter |
| Yes | No | Yes | Yes | Yes |

Control/Data Signals

| Signal | User Programmable | Use |
|--------------------------|----------------------|--|
| SSF | No | Affect shift instructions programmed in bits I7-I4 of instruction field. |
| $\overline{\text{SIO0}}$ | No | |
| $\overline{\text{SIO1}}$ | No | |
| $\overline{\text{SIO2}}$ | No | |
| $\overline{\text{SIO3}}$ | No | |
| Cn | Yes | Two's complement subtraction if programmed high. |

Status Signals†

ZERO = 1 if result = 0
 N = 1 if MSB = 1
 OVR = 1 if signed arithmetic overflow
 C = 1 if carry-out

†C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Subtract data in register 1 from data on the DB bus, and store the result in the MQ register.

| Instr Code I7-I0 | Oprd Addr A5-A0 | Oprd Addr B5-B0 | Oprd Sel EB1- EA EB0 | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2- CF0 |
|------------------------|-----------------------|-----------------------|----------------------------|-----------------------|---------------------|-------------|-------------------|-----|-----|-------------|-----|---|-----|-------------|
| | | | | | SELMQ | WE3- WE0 | SELRF1- SELRF0 | OEA | OEB | OY3- OY0 | OES | | | |
| 1110 0010 | 00 0001 | XX XXXX | 0 10 | XX XXXX | 1 | XXXX | XX | X | X | XXXX | 0 | 1 | 110 | |

Assume register file 1 holds 150084D0 (Hex) and DB bus holds 4900C350 (Hex).

Source 0001 0101 0000 0000 1000 0100 1101 0000 R ← RF(1)

Source 0100 1001 0000 0000 1100 0011 0101 0000 S ← DB bus

Destination 0011 0100 0000 0000 0011 1110 1000 0000 MQ register ← $R' + S + C_n$

SUBS**Subtract S with Carry (R + S' + Cn)***** 3****FUNCTION**

Subtracts data on the S bus from R with carry.

DESCRIPTION

Data on the S bus is subtracted with carry from data on the R bus. The result appears at the ALU and MQ shifters.

*The result of this instruction can be shifted in the same microcycle by specifying a shift instruction in the upper nibble (I7-I4) of the instruction field. The result may also be passed without shift. Possible instructions are listed in Table 15.

Available R Bus Source Operands

| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 .. A3-A0 Mask |
|---------------|----------------|---------|------------------------------|
| Yes | No | Yes | No |

Available S Bus Source Operands

| RF (B5-B0) | DB-Port | MQ Register |
|---------------|---------|----------------|
| Yes | Yes | Yes |

Available Destination Operands

| RF (C5-C0) | RF (B5-B0) | Y-Port | ALU Shifter | MQ Shifter |
|---------------|---------------|--------|----------------|---------------|
| Yes | No | Yes | Yes | Yes |

Control/Data Signals

| Signal | User Programmable | Use |
|--------------------------|----------------------|--|
| SSF | No | Affect shift instructions programmed in bits I7-I4 of instruction field. |
| $\overline{\text{SIO0}}$ | No | |
| $\overline{\text{SIO1}}$ | No | |
| $\overline{\text{SIO2}}$ | No | |
| $\overline{\text{SIO3}}$ | No | |
| Cn | Yes | Two's complement subtraction if programmed high. |

Status Signals[†]

| | |
|------|-----------------------------------|
| ZERO | = 1 if result = 0 |
| N | = 1 if MSB = 1 |
| OVR | = 1 if signed arithmetic overflow |
| C | = 1 if carry-out |

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Subtract data on the DB bus from data in register 1, and store the result in the MQ register.

| Instr Code | Oprd Addr | Oprd Addr | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2- CF0 |
|---------------|--------------|--------------|----------------------------|-----------------------|---------------------|-------------|-------------------|-------------|---------------|------|-----|---|-----|-------------|
| | | | | | SEL | WE3- WE0 | SELRF1- SELRF0 | OE3- OE0 | OEY3- OEY0 | OES | | | | |
| 17-10 | A5-A0 | B5-B0 | EA EBO | C5-C0 | SEL | WE0 | SELRF0 | OE3 | OE0 | OEY0 | OES | | | |
| 1110 0011 | 00 0001 | XX XXXX | 0 10 | XX XXXX | 1 | XXXX | XX | X | X | XXXX | 0 | 1 | 110 | |

Assume register file 1 holds 150084D0 (Hex) and DB bus holds 4900C350 (Hex).

| | | |
|-------------|---|---------------------------|
| Source | 0001 0101 0000 0000 1000 0100 1101 0000 | R ← RF(1) |
| Source | 0100 1001 0000 0000 1100 0011 0101 0000 | S ← DB bus |
| Destination | 1100 1011 1111 1111 1100 0001 1000 0000 | MQ register ← R + S' + Cn |

| Signal | User Programmable | Use |
|--------|-------------------|-------------|
| Cn | No | Inactive |
| St03 | Yes | Byte Select |
| St02 | Yes | Byte Select |
| St01 | Yes | Byte Select |
| St00 | Yes | Byte Select |
| St04 | Yes | Byte Select |
| St05 | Yes | Byte Select |
| St06 | Yes | Byte Select |
| St07 | Yes | Byte Select |
| St08 | Yes | Byte Select |
| St09 | Yes | Byte Select |
| St10 | Yes | Byte Select |
| St11 | Yes | Byte Select |
| St12 | Yes | Byte Select |
| St13 | Yes | Byte Select |
| St14 | Yes | Byte Select |
| St15 | Yes | Byte Select |
| St16 | Yes | Byte Select |
| St17 | Yes | Byte Select |
| St18 | Yes | Byte Select |
| St19 | Yes | Byte Select |
| St20 | Yes | Byte Select |
| St21 | Yes | Byte Select |
| St22 | Yes | Byte Select |
| St23 | Yes | Byte Select |
| St24 | Yes | Byte Select |
| St25 | Yes | Byte Select |
| St26 | Yes | Byte Select |
| St27 | Yes | Byte Select |
| St28 | Yes | Byte Select |
| St29 | Yes | Byte Select |
| St30 | Yes | Byte Select |
| St31 | Yes | Byte Select |
| St32 | Yes | Byte Select |
| St33 | Yes | Byte Select |
| St34 | Yes | Byte Select |
| St35 | Yes | Byte Select |
| St36 | Yes | Byte Select |
| St37 | Yes | Byte Select |
| St38 | Yes | Byte Select |
| St39 | Yes | Byte Select |
| St40 | Yes | Byte Select |
| St41 | Yes | Byte Select |
| St42 | Yes | Byte Select |
| St43 | Yes | Byte Select |
| St44 | Yes | Byte Select |
| St45 | Yes | Byte Select |
| St46 | Yes | Byte Select |
| St47 | Yes | Byte Select |
| St48 | Yes | Byte Select |
| St49 | Yes | Byte Select |
| St50 | Yes | Byte Select |
| St51 | Yes | Byte Select |
| St52 | Yes | Byte Select |
| St53 | Yes | Byte Select |
| St54 | Yes | Byte Select |
| St55 | Yes | Byte Select |
| St56 | Yes | Byte Select |
| St57 | Yes | Byte Select |
| St58 | Yes | Byte Select |
| St59 | Yes | Byte Select |
| St60 | Yes | Byte Select |
| St61 | Yes | Byte Select |
| St62 | Yes | Byte Select |
| St63 | Yes | Byte Select |
| St64 | Yes | Byte Select |
| St65 | Yes | Byte Select |
| St66 | Yes | Byte Select |
| St67 | Yes | Byte Select |
| St68 | Yes | Byte Select |
| St69 | Yes | Byte Select |
| St70 | Yes | Byte Select |
| St71 | Yes | Byte Select |
| St72 | Yes | Byte Select |
| St73 | Yes | Byte Select |
| St74 | Yes | Byte Select |
| St75 | Yes | Byte Select |
| St76 | Yes | Byte Select |
| St77 | Yes | Byte Select |
| St78 | Yes | Byte Select |
| St79 | Yes | Byte Select |
| St80 | Yes | Byte Select |
| St81 | Yes | Byte Select |
| St82 | Yes | Byte Select |
| St83 | Yes | Byte Select |
| St84 | Yes | Byte Select |
| St85 | Yes | Byte Select |
| St86 | Yes | Byte Select |
| St87 | Yes | Byte Select |
| St88 | Yes | Byte Select |
| St89 | Yes | Byte Select |
| St90 | Yes | Byte Select |
| St91 | Yes | Byte Select |
| St92 | Yes | Byte Select |
| St93 | Yes | Byte Select |
| St94 | Yes | Byte Select |
| St95 | Yes | Byte Select |
| St96 | Yes | Byte Select |
| St97 | Yes | Byte Select |
| St98 | Yes | Byte Select |
| St99 | Yes | Byte Select |
| St100 | Yes | Byte Select |
| St101 | Yes | Byte Select |
| St102 | Yes | Byte Select |
| St103 | Yes | Byte Select |
| St104 | Yes | Byte Select |
| St105 | Yes | Byte Select |
| St106 | Yes | Byte Select |
| St107 | Yes | Byte Select |
| St108 | Yes | Byte Select |
| St109 | Yes | Byte Select |
| St110 | Yes | Byte Select |
| St111 | Yes | Byte Select |
| St112 | Yes | Byte Select |
| St113 | Yes | Byte Select |
| St114 | Yes | Byte Select |
| St115 | Yes | Byte Select |
| St116 | Yes | Byte Select |
| St117 | Yes | Byte Select |
| St118 | Yes | Byte Select |
| St119 | Yes | Byte Select |
| St120 | Yes | Byte Select |
| St121 | Yes | Byte Select |
| St122 | Yes | Byte Select |
| St123 | Yes | Byte Select |
| St124 | Yes | Byte Select |
| St125 | Yes | Byte Select |
| St126 | Yes | Byte Select |
| St127 | Yes | Byte Select |
| St128 | Yes | Byte Select |
| St129 | Yes | Byte Select |
| St130 | Yes | Byte Select |
| St131 | Yes | Byte Select |
| St132 | Yes | Byte Select |
| St133 | Yes | Byte Select |
| St134 | Yes | Byte Select |
| St135 | Yes | Byte Select |
| St136 | Yes | Byte Select |
| St137 | Yes | Byte Select |
| St138 | Yes | Byte Select |
| St139 | Yes | Byte Select |
| St140 | Yes | Byte Select |
| St141 | Yes | Byte Select |
| St142 | Yes | Byte Select |
| St143 | Yes | Byte Select |
| St144 | Yes | Byte Select |
| St145 | Yes | Byte Select |
| St146 | Yes | Byte Select |
| St147 | Yes | Byte Select |
| St148 | Yes | Byte Select |
| St149 | Yes | Byte Select |
| St150 | Yes | Byte Select |
| St151 | Yes | Byte Select |
| St152 | Yes | Byte Select |
| St153 | Yes | Byte Select |
| St154 | Yes | Byte Select |
| St155 | Yes | Byte Select |
| St156 | Yes | Byte Select |
| St157 | Yes | Byte Select |
| St158 | Yes | Byte Select |
| St159 | Yes | Byte Select |
| St160 | Yes | Byte Select |
| St161 | Yes | Byte Select |
| St162 | Yes | Byte Select |
| St163 | Yes | Byte Select |
| St164 | Yes | Byte Select |
| St165 | Yes | Byte Select |
| St166 | Yes | Byte Select |
| St167 | Yes | Byte Select |
| St168 | Yes | Byte Select |
| St169 | Yes | Byte Select |
| St170 | Yes | Byte Select |
| St171 | Yes | Byte Select |
| St172 | Yes | Byte Select |
| St173 | Yes | Byte Select |
| St174 | Yes | Byte Select |
| St175 | Yes | Byte Select |
| St176 | Yes | Byte Select |
| St177 | Yes | Byte Select |
| St178 | Yes | Byte Select |
| St179 | Yes | Byte Select |
| St180 | Yes | Byte Select |
| St181 | Yes | Byte Select |
| St182 | Yes | Byte Select |
| St183 | Yes | Byte Select |
| St184 | Yes | Byte Select |
| St185 | Yes | Byte Select |
| St186 | Yes | Byte Select |
| St187 | Yes | Byte Select |
| St188 | Yes | Byte Select |
| St189 | Yes | Byte Select |
| St190 | Yes | Byte Select |
| St191 | Yes | Byte Select |
| St192 | Yes | Byte Select |
| St193 | Yes | Byte Select |
| St194 | Yes | Byte Select |
| St195 | Yes | Byte Select |
| St196 | Yes | Byte Select |
| St197 | Yes | Byte Select |
| St198 | Yes | Byte Select |
| St199 | Yes | Byte Select |
| St200 | Yes | Byte Select |
| St201 | Yes | Byte Select |
| St202 | Yes | Byte Select |
| St203 | Yes | Byte Select |
| St204 | Yes | Byte Select |
| St205 | Yes | Byte Select |
| St206 | Yes | Byte Select |
| St207 | Yes | Byte Select |
| St208 | Yes | Byte Select |
| St209 | Yes | Byte Select |
| St210 | Yes | Byte Select |
| St211 | Yes | Byte Select |
| St212 | Yes | Byte Select |
| St213 | Yes | Byte Select |
| St214 | Yes | Byte Select |
| St215 | Yes | Byte Select |
| St216 | Yes | Byte Select |
| St217 | Yes | Byte Select |
| St218 | Yes | Byte Select |
| St219 | Yes | Byte Select |
| St220 | Yes | Byte Select |
| St221 | Yes | Byte Select |
| St222 | Yes | Byte Select |
| St223 | Yes | Byte Select |
| St224 | Yes | Byte Select |
| St225 | Yes | Byte Select |
| St226 | Yes | Byte Select |
| St227 | Yes | Byte Select |
| St228 | Yes | Byte Select |
| St229 | Yes | Byte Select |
| St230 | Yes | Byte Select |
| St231 | Yes | Byte Select |
| St232 | Yes | Byte Select |
| St233 | Yes | Byte Select |
| St234 | Yes | Byte Select |
| St235 | Yes | Byte Select |
| St236 | Yes | Byte Select |
| St237 | Yes | Byte Select |
| St238 | Yes | Byte Select |
| St239 | Yes | Byte Select |
| St240 | Yes | Byte Select |
| St241 | Yes | Byte Select |
| St242 | Yes | Byte Select |
| St243 | Yes | Byte Select |
| St244 | Yes | Byte Select |
| St245 | Yes | Byte Select |
| St246 | Yes | Byte Select |
| St247 | Yes | Byte Select |
| St248 | Yes | Byte Select |
| St249 | Yes | Byte Select |
| St250 | Yes | Byte Select |
| St251 | Yes | Byte Select |
| St252 | Yes | Byte Select |
| St253 | Yes | Byte Select |
| St254 | Yes | Byte Select |
| St255 | Yes | Byte Select |
| St256 | Yes | Byte Select |
| St257 | Yes | Byte Select |
| St258 | Yes | Byte Select |
| St259 | Yes | Byte Select |
| St260 | Yes | Byte Select |
| St261 | Yes | Byte Select |
| St262 | Yes | Byte Select |
| St263 | Yes | Byte Select |
| St264 | Yes | Byte Select |
| St265 | Yes | Byte Select |
| St266 | Yes | Byte Select |
| St267 | Yes | Byte Select |
| St268 | Yes | Byte Select |
| St269 | Yes | Byte Select |
| St270 | Yes | Byte Select |
| St271 | Yes | Byte Select |
| St272 | Yes | Byte Select |
| St273 | Yes | Byte Select |
| St274 | Yes | Byte Select |
| St275 | Yes | Byte Select |
| St276 | Yes | Byte Select |
| St277 | Yes | Byte Select |
| St278 | Yes | Byte Select |
| St279 | Yes | Byte Select |
| St280 | Yes | Byte Select |
| St281 | Yes | Byte Select |
| St282 | Yes | Byte Select |
| St283 | Yes | Byte Select |
| St284 | Yes | Byte Select |
| St285 | Yes | Byte Select |
| St286 | Yes | Byte Select |
| St287 | Yes | Byte Select |
| St288 | Yes | Byte Select |
| St289 | Yes | Byte Select |
| St290 | Yes | Byte Select |
| St291 | Yes | Byte Select |
| St292 | Yes | Byte Select |
| St293 | Yes | Byte Select |
| St294 | Yes | Byte Select |
| St295 | Yes | Byte Select |
| St296 | Yes | Byte Select |
| St297 | Yes | Byte Select |
| St298 | Yes | Byte Select |
| St299 | Yes | Byte Select |
| St300 | Yes | Byte Select |
| St301 | Yes | Byte Select |
| St302 | Yes | Byte Select |
| St303 | Yes | Byte Select |
| St304 | Yes | Byte Select |
| St305 | Yes | Byte Select |
| St306 | Yes | Byte Select |
| St307 | Yes | Byte Select |
| St308 | Yes | Byte Select |
| St309 | Yes | Byte Select |
| St310 | Yes | Byte Select |
| St311 | Yes | Byte Select |
| St312 | Yes | Byte Select |
| St313 | Yes | Byte Select |
| St314 | Yes | Byte Select |
| St315 | Yes | Byte Select |
| St316 | Yes | Byte Select |
| St317 | Yes | Byte Select |
| St318 | Yes | Byte Select |
| St319 | Yes | Byte Select |
| St320 | Yes | Byte Select |
| St321 | Yes | Byte Select |
| St322 | Yes | Byte Select |
| St323 | Yes | Byte Select |
| St324 | Yes | Byte Select |
| St325 | Yes | Byte Select |
| St326 | Yes | Byte Select |
| St327 | Yes | Byte Select |
| St328 | Yes | Byte Select |
| St329 | Yes | Byte Select |
| St330 | Yes | Byte Select |
| St331 | Yes | Byte Select |
| St332 | Yes | Byte Select |
| St333 | Yes | Byte Select |
| St334 | Yes | Byte Select |
| St335 | Yes | Byte Select |
| St336 | Yes | Byte Select |
| St337 | Yes | Byte Select |
| St338 | Yes | Byte Select |
| St339 | Yes | Byte Select |
| St340 | Yes | Byte Select |
| St341 | Yes | Byte Select |
| St342 | Yes | Byte Select |
| St343 | Yes | Byte Select |
| St344 | Yes | Byte Select |
| St345 | Yes | Byte Select |
| St346 | Yes | Byte Select |
| St347 | Yes | Byte Select |
| St348 | Yes | Byte Select |
| St349 | Yes | Byte Select |
| St350 | Yes | Byte Select |
| St351 | Yes | Byte Select |
| St352 | Yes | Byte Select |
| St353 | Yes | Byte Select |
| St354 | Yes | Byte Select |
| St355 | Yes | Byte Select |
| St356 | Yes | Byte Select |
| St357 | Yes | Byte Select |
| St358 | Yes | Byte Select |
| St359 | Yes | Byte Select |
| St360 | Yes | Byte Select |
| St361 | Yes | Byte Select |
| St362 | Yes | Byte Select |
| St363 | Yes | Byte Select |
| St364 | Yes | Byte Select |
| St365 | Yes | Byte Select |
| St366 | Yes | Byte Select |
| St367 | Yes | Byte Select |
| St368 | Yes | Byte Select |
| St369 | Yes | Byte Select |
| St370 | Yes | Byte Select |
| St371 | Yes | Byte Select |
| St372 | Yes | Byte Select |
| St373 | Yes | Byte Select |
| St374 | Yes | Byte Select |
| St375 | Yes | Byte Select |
| St376 | Yes | Byte Select |
| St377 | Yes | Byte Select |
| St378 | Yes | Byte Select |
| St379 | Yes | Byte Select |
| St380 | Yes | Byte Select |
| St381 | Yes | Byte Select |
| St382 | Yes | Byte Select |
| St383 | Yes | Byte Select |
| St384 | Yes | Byte Select |
| St385 | Yes | Byte Select |
| St386 | Yes | Byte Select |
| St387 | Yes | Byte Select |
| St388 | Yes | Byte Select |
| St389 | Yes | Byte Select |
| St390 | Yes | Byte Select |
| St391 | Yes | Byte Select |
| St392 | Yes | Byte Select |
| St393 | Yes | Byte Select |
| St394 | Yes | Byte Select |
| St395 | Yes | Byte Select |

FUNCTION

Tests bits in selected bytes of S-bus data for zeros using mask in C3-C0::A3-A0.

DESCRIPTION

The S bus is the source word for this instruction. The source word is passed to the ALU, where it is compared to an 8-bit mask, consisting of a concatenation of the C3-C0 and A3-A0 address ports (C3-C0::A3-A0). The mask is input via the R bus. The test will pass if the selected byte has zeros at all bit locations specified by the ones of the mask. Bytes are selected by programming the \overline{SIO} inputs low. Test results are indicated on the ZERO output, which goes to one if the test passes. Register write is internally disabled during this instruction.

Available R Bus Source Operands

| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
|---------------|----------------|---------|------------------------------|
| No | No | No | Yes |

Available S Bus Source Operands

| RF (B5-B0) | DB-Port | MQ Register |
|---------------|---------|----------------|
| Yes | Yes | Yes |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|----------------------|-------------|
| SSF | No | Inactive |
| $\overline{SIO0}$ | Yes | Byte Select |
| $\overline{SIO1}$ | Yes | Byte Select |
| $\overline{SIO2}$ | Yes | Byte Select |
| $\overline{SIO3}$ | Yes | Byte Select |
| Cn | No | Inactive |

Status Signals

ZERO = 1 if result (selected bytes) = Pass
N = 0
OVR = 0
C = 0

EXAMPLE (assumes a 32-bit configuration)

Test bits 7, 6 and 5 of bytes 0 and 2 of data in register 3 for zeroes.

| Instr Code | Mask (LSH) | Oprd Addr | Oprd Sel EB1- EA EBO | Mask (MSH) C3-C0 | Destination Selects | | | | | | | | Cn | CF2- CF0 | SIO3- SIO0 | IESIO3- IESIO0 |
|---------------|---------------|--------------|----------------------------|------------------------|---------------------|-------------------|---------------|-----|-----|------|-----|-----|-----|-------------|---------------|-------------------|
| | | | | | WE3- SELWQ | SELRF1- SELRF0 | OEY3- OEY0 | OES | OEA | OEB | OES | OES | | | | |
| I7-I0 | A3-A0 | B5-B0 | EA EBO | C3-C0 | SELWQ | WE0 | SELRF0 | OEA | OEB | OEY0 | OES | Cn | CF0 | SIO0 | IESIO0 | |
| 0011 1000 | 0000 | 00 0011 | X 00 | 1110 | X | XXXX | XX | X | X | XXXX | 0 | X | 110 | 1010 | 0000 | |

Assume register file 3 holds 881CD003 (Hex).

Source 1110 0000 1110 0000 1110 0000 1110 0000 R ← Mask (C3-C0::A3-A0)

Source 1000 1000 0001 1100 1101 0000 0000 0011 SN ← RF(3)n†

Output 1 ZERO ← 1

†n = nth byte

| Signal | Programmable | Use |
|--------|--------------|-------------|
| Cn | No | Inactive |
| St03 | Yes | Byte Select |
| St02 | Yes | Byte Select |
| St01 | Yes | Byte Select |
| St00 | Yes | Byte Select |
| Szf | No | Inactive |

FUNCTION

Tests bits in selected bytes of S-bus data for ones using mask in C3-C0::A3-A0.

DESCRIPTION

The S bus is the source word for this instruction. The source word is passed to the ALU, where it is compared to an 8-bit mask, consisting of a concatenation of the C3-C0 and A3-A0 address ports (C3-C0::A3-A0). The mask is input via the R bus. The test will pass if the selected byte has ones at all bit locations specified by the ones of the mask. Bytes are selected by programming the \overline{SIO} inputs low. Test results are indicated on the ZERO output, which goes to one if the test passes. Register write is internally disabled for this instruction.

Available R Bus Source Operands

| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
|---------------|----------------|---------|------------------------------|
| No | No | No | Yes |

Available S Bus Source Operands

| RF (B5-B0) | DB-Port | MQ Register |
|---------------|---------|----------------|
| Yes | Yes | Yes |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|----------------------|-------------|
| SSF | No | Inactive |
| $\overline{SIO0}$ | Yes | Byte Select |
| $\overline{SIO1}$ | Yes | Byte Select |
| $\overline{SIO2}$ | Yes | Byte Select |
| $\overline{SIO3}$ | Yes | Byte Select |
| Cn | No | Inactive |

Status Signals

| | |
|------|---------------------------------------|
| ZERO | = 1 if result (selected bytes) = Pass |
| N | = 0 |
| OVR | = 0 |
| C | = 0 |

EXAMPLE (assumes a 32-bit configuration)

Test bits 7, 6 and 5 of bytes 1 and 2 of data in register 3 for ones.

| Instr Code | Mask (LSH) | Oprd Addr | Oprd Sel EB1- EA EBO | Mask (MSH) C3-C0 | Destination Selects | | | | | | | | Cn | CF2- CF0 | SIO3- SIO0 | IESIO3- IESIO0 |
|---------------|---------------|--------------|----------------------------|------------------------|---------------------|---------|--------|-------|-----|------|-----|------|----|-------------|---------------|-------------------|
| | | | | | WE3- | SELRF1- | OEY3- | OEY0- | OES | OEY0 | OES | OEY0 | | | | |
| 17-10 | A3-A0 | B5-B0 | EA EBO | C3-C0 | SELMO | WE0 | SELRF0 | OEA | OEB | OEY0 | OES | OEY0 | X | 110 | 1001 | 0000 |
| 0010 1000 | 0000 | 00 0011 | X 00 | 1110 | X | XXXX | XX | X | X | XXXX | 0 | X | X | 110 | 1001 | 0000 |

Assume register file 3 holds 881CF003 (Hex).

Mask 1110 0000 1110 0000 1110 0000 1110 0000 Rn ← Mask (C3-C0::A3-A0)

Source 1000 1000 0001 1100 1101 0000 0000 0011 Sn ← RF(3)n†

Output 0 ZERO ← 0

†n = nth byte

FUNCTION

Performs one of N-2 iterations of nonrestoring unsigned division by a test subtraction of the N-bit divisor from the 2N-bit dividend. An algorithm using this instruction can be found in the "Other Arithmetic Instructions" section.

DESCRIPTION

UDIVI performs a test subtraction of the divisor from the dividend to generate a quotient bit. The test subtraction may pass or fail and is corrected in the subsequent instruction if it fails. Similarly a failed test from the previous instruction is corrected during evaluation of the current UDIVI instruction (see the "Other Arithmetic Instructions" section for more details).

The R bus must be loaded with the divisor, the S bus with the most significant half of the result of the previous instruction (UDIVI during iteration or UDIVIS at the beginning of iteration). The least significant half of the previous result is in the MQ register.

UDIVI checks the result of the previous pass/fail test and then evaluates:

$$\begin{aligned} F &\leftarrow R + S && \text{if the test is failed} \\ F &\leftarrow R' + S + C_n && \text{if the test is passed} \end{aligned}$$

A double precision left shift is performed; bit 7 of the most significant byte of the MQ shifter is transferred to bit 0 of the least significant byte of the ALU shifter. Bit 7 of the most significant byte of the ALU shifter is lost. The unfixed quotient bit is circulated into the least significant bit of the MQ shifter.

Available R Bus Source Operands

| | | | |
|---------------|----------------|---------|------------------------------|
| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
| Yes | No | Yes | No |

Recommended S Bus Source Operands

| | | |
|---------------|---------|----------------|
| RF (B5-B0) | DB-Port | MQ Register |
| Yes | Yes | No |

Recommended Destination Operands Shift Operations

| | | | | |
|---------------|---------------|--------|------|------|
| RF (C5-C0) | RF (B5-B0) | Y-Port | ALU | MQ |
| Yes | No | Yes | Left | Left |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|-------------------|---|
| SSF | No | Inactive |
| $\overline{STO0}$ | No | Passes internally generated end-fill bit. |
| $\overline{STO1}$ | No | |
| $\overline{STO2}$ | No | |
| $\overline{STO3}$ | No | |
| Cn | Yes | Should be programmed high. |

Status Signals

| | |
|------|-------------------|
| ZERO | = 1 if result = 0 |
| N | = 0 |
| OVR | = 0 |
| C | = 1 if carry-out |

FUNCTION

Computes the first quotient bit of nonrestoring unsigned division. An algorithm using this instruction is given in the "Other Arithmetic Instructions" section.

DESCRIPTION

UDIVIS computes the first quotient bit during nonrestoring unsigned division by subtracting the divisor from the dividend. The resulting remainder due to subtraction may be negative; the subsequent UDIVI instruction may have to restore the remainder during the next operation.

The R bus must be loaded with the divisor and the S bus with the most significant half of the remainder. The result on the Y bus should be loaded back into the register file for use in the next instruction. The least significant half of the remainder is in the MQ register.

UDIVIS computes:

$$F \leftarrow R' + S + Cn$$

A double precision left shift is performed; bit 7 of the most significant byte of the MQ shifter is transferred to bit 0 of the least significant byte of the ALU shifter. Bit 7 of the most significant byte of the ALU shifter is lost. The unfixed quotient bit is circulated into the least significant bit of the MQ shifter.

Available R Bus Source Operands

| | | | |
|---------------|----------------|---------|------------------------------|
| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
| Yes | No | Yes | No |

Recommended S Bus Source Operands

| | | |
|---------------|---------|----------------|
| RF (B5-B0) | DB-Port | MQ Register |
| Yes | Yes | No |

Recommended Destination Operands Shift Operations

| | | |
|---------------|---------------|--------|
| RF (C5-C0) | RF (B5-B0) | Y-Port |
| Yes | No | Yes |

| | |
|------|------|
| ALU | MQ |
| Left | Left |

Control/Data Signals

| Signal | User Programmable | Use |
|--------------------------|-------------------|---|
| SSF | No | Inactive |
| $\overline{\text{SIO0}}$ | No | Passes internally generated end-fill bit. |
| $\overline{\text{SIO1}}$ | No | |
| $\overline{\text{SIO2}}$ | No | |
| $\overline{\text{SIO3}}$ | No | |
| Cn | Yes | Should be programmed high. |

Status Signals

| | |
|------|--------------------------------|
| ZERO | = 1 if intermediate result = 0 |
| N | = 0 |
| OVR | = 1 if divide overflow |
| C | = 1 if carry-out |

FUNCTION

Solves the final quotient bit during nonrestoring unsigned division. An algorithm using this instruction is given in the "Other Arithmetic Instructions" section.

DESCRIPTION

UDIVIT performs the final subtraction of the divisor from the remainder during nonrestoring signed division. UDIVIT is preceded by N-1 iterations of UDIVI, where N is the number of bits in the dividend.

The R bus must be loaded with the divisor, the S bus must be loaded with the most significant half of the result of the last UDIVI instruction. The least significant half lies in the MQ register. The Y bus result must be loaded back into the register file for use in the subsequent DIVRF instruction.

UDIVIT checks the results of the previous pass/fail test and evaluates:

$$\begin{aligned} Y &\leftarrow R + S && \text{if the test is failed} \\ Y &\leftarrow R' + S + C_n && \text{if the test is passed} \end{aligned}$$

The contents of the MQ register are shifted one bit to the left; the unfixed quotient bit is circulated into the least significant bit.

Available R Bus Source Operands

| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
|---------------|----------------|---------|------------------------------|
| Yes | No | Yes | No |

Recommended S Bus Source Operands

| RF (B5-B0) | DB-Port | MQ Register |
|---------------|---------|----------------|
| Yes | Yes | No |

Recommended Destination Operands Shift Operations

| RF (C5-C0) | RF (B5-B0) | Y-Port |
|---------------|---------------|--------|
| Yes | No | Yes |

| ALU | MQ |
|------|------|
| None | Left |

Control/Data Signals

| Signal | User Programmable | Use |
|--------------------------|-------------------|---|
| SSF | No | Inactive |
| $\overline{\text{STO0}}$ | No | Passes internally generated end-fill bit. |
| $\overline{\text{STO1}}$ | No | |
| $\overline{\text{STO2}}$ | No | |
| $\overline{\text{STO3}}$ | No | |
| Cn | Yes | Should be programmed high. |

Status Signals

| | |
|------|--------------------------------|
| ZERO | = 1 if intermediate result = 0 |
| N | = 0 |
| OVR | = 0 |
| C | = 1 if carry-out |

FUNCTION

Performs one of N unsigned multiplication iterations for computing an N-bit by N-bit product. An algorithm for unsigned multiplication using this instruction is given in the "Other Arithmetic Instructions" section.

DESCRIPTION

UMULI checks to determine whether the multiplicand should be added with the present partial product. The instruction evaluates:

$F \leftarrow R + S + C_n$ if the addition is required
 $F \leftarrow S$ if no addition is required

A double precision right shift is performed. Bit 0 of the least significant byte of the ALU shifter is passed to bit 7 of the most significant byte of the MQ shifter; carry-out is passed to the most significant bit of the ALU shifter.

The S bus should be loaded with the contents of an accumulator and the R bus with the multiplicand. The Y bus result should be written back to the accumulator after each iteration of UMULI. The accumulator should be cleared and the MQ register loaded with the multiplier before the first iteration.

R Bus Source Operands

| | | | |
|---------------|----------------|---------|------------------------------|
| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
| Yes | No | Yes | No |

Recommended S Bus Source Operands

| | | |
|---------------|---------|----------------|
| RF (B5-B0) | DB-Port | MQ Register |
| Yes | Yes | No |

Recommended Destination Operands Shift Operations

| | | |
|---------------|---------------|--------|
| RF (C5-C0) | RF (B5-B0) | Y-Port |
| Yes | No | Yes |

| | |
|-------|-------|
| ALU | MQ |
| Right | Right |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|-------------------|--------------------------------------|
| SSF | No | Holds LSB of MQ. |
| $\overline{SIO0}$ | No | Passes internal input (shifted bit). |
| $\overline{SIO1}$ | No | |
| $\overline{SIO2}$ | No | |
| $\overline{SIO3}$ | No | |
| Cn | Yes | Should be programmed low. |

Status Signals[†]

| |
|------------------------|
| ZERO = 1 if result = 0 |
| N = 1 if MSB = 1 |
| OVR = 0 |
| C = 1 if carry-out |

[†]Valid only on final execution of multiply iteration

FUNCTION

Evaluates the logical expression R XOR S.

DESCRIPTION

Data on the R bus is exclusive ORed with data on the S bus. The result appears at the ALU and MQ shifters.

*The result of this instruction can be shifted in the same microcycle by specifying a shift instruction in the upper nibble (I7-I4) of the instruction field. The result may also be passed without shift. Possible instructions are listed in Table 15.

Available R Bus Source Operands

| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
|---------------|----------------|---------|------------------------------|
| Yes | No | Yes | No |

Available S Bus Source Operands

| RF (B5-B0) | DB-Port | MQ Register |
|---------------|---------|----------------|
| Yes | Yes | Yes |

Available Destination Operands

| RF (C5-C0) | RF (B5-B0) | Y-Port | ALU Shifter | MQ Shifter |
|---------------|---------------|--------|----------------|---------------|
| Yes | No | Yes | Yes | Yes |

Control/Data Signals

| Signal | User Programmable | Use |
|--------------------------|----------------------|--|
| SSF | No | Affect shift instructions programmed in bits I7-I4 of instruction field. |
| $\overline{\text{SIO0}}$ | No | |
| $\overline{\text{SIO1}}$ | No | |
| $\overline{\text{SIO2}}$ | No | |
| $\overline{\text{SIO3}}$ | No | |
| Cn | No | Inactive |

Status Signals[†]

ZERO = 1 if result = 0
 N = 1 if MSB = 1
 OVR = 0
 C = 0

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Exclusive OR the contents of register 3 and register 5, and store the result in register 5.

| Instr Code | Oprd Addr | Oprd Addr | Oprd Sel EB1- EA EB0 | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2- CF0 |
|---------------|--------------|--------------|----------------------------|-----------------------|---------------------|-------------------|------------|--------------|------------|--------------|------------|--------------|----|-------------|
| | | | | | WE3- WE0 | SELRF1- SELRF0 | OE3 OE0 | OEY3 OEY0 | OE3 OE0 | OEY3 OEY0 | OE3 OE0 | OEY3 OEY0 | | |
| 17-10 | A5-A0 | B5-B0 | EA EB0 | C5-C0 | SELMO | WE0 | SELRF0 | OE3 | OE0 | OEY3 | OEY0 | OE3 | X | 110 |
| 1111 1001 | 00 0011 | 00 0101 | 0 00 | 00 0101 | 0 | 0000 | 10 | X | X | XXXX | 0 | X | X | 110 |

Assume register file 3 holds 33F6D840 (Hex) and register file 5 holds 90F6D842 (Hex)..

Source 0011 0011 1111 0110 1101 1000 0100 0000 R ← RF(3)

Source 1001 0000 1111 0110 1101 1000 0100 0010 S ← RF(5)

Destination 1010 0011 0000 0000 0000 0000 0000 0010 RF(5) ← R XOR S

Status Signals

| | |
|------|---|
| ZERO | = 1 if result (selected bytes) = 0 |
| N | = 0 |
| OVR | = 1 if signed arithmetic overflow (selected bytes) |
| C | = 1 if carry-out (most significant selected byte) = 1 |

EXAMPLE (assumes a 32-bit configuration)

Add bytes 1 and 2 of register 3 with carry to the contents of register 1 and store the result in register 11.

| Instr Code | Oprd Addr | Oprd Addr | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2- CF0 | SIO3- SIO0 | IESIO3- IESIO0 |
|---------------|--------------|--------------|----------------------------|-----------------------|---------------------|-------------|-------------------|-----|-----|---------------|-----|---|-----|-------------|---------------|-------------------|
| | | | | | SELMO | WE3- WE0 | SELRF1- SELRF0 | OEA | OEB | OEY3- OEY0 | OES | | | | | |
| 0100 1000 | 00 0011 | 00 0001 | 0 00 | 00 1011 | 0 | 0000 | 10 | X | X | XXXX | 0 | 1 | 110 | 1001 | 0000 | |

Assume register file 3 holds 2C018181 (Hex) and register file 1 holds 7A8FBE3E (Hex):

Source 0010 1100 0000 0001 1000 0001 1000 0001 $R_n \leftarrow RF(3)n$

Source 0111 1010 1000 1111 1011 1110 0011 1110 $S_n \leftarrow RF(1)n$

ALU 1010 0110 1001 0001 0100 0000 1100 0000 $F_n \leftarrow R_n + S_n + C_n$

Destination 0111 1010 1001 0001 0100 1111 0011 1110 $RF(11)n \leftarrow F_n \text{ or } S_n^\dagger$

$^\dagger F$ = ALU result

n = nth byte

Register file 11 gets F if byte selected, S if byte not selected.

FUNCTION

Evaluates the logical AND of selected bytes of R-bus and S-bus data.

DESCRIPTION

Bytes with their corresponding \overline{SIO} signals programmed low compute R AND S. Bytes with \overline{SIO} signals programmed high, pass S unaltered. Multiple bytes can be selected only if they are adjacent to one another. At least one byte must be nonselected.

Available R Bus Source Operands

| | | | |
|---------------|----------------|---------|------------------------------|
| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
| Yes | No | Yes | No |

Available S Bus Source Operands

| | | |
|---------------|---------|----------------|
| RF (B5-B0) | DB-Port | MQ Register |
| Yes | Yes | Yes |

Available Destination Operands Shift Operations

| | | | | |
|---------------|---------------|--------|------|------|
| RF (C5-C0) | RF (B5-B0) | Y-Port | ALU | MQ |
| Yes | No | Yes | None | None |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|-------------------|-------------|
| SSF | No | Forced low |
| $\overline{SIO0}$ | Yes | Byte select |
| $\overline{SIO1}$ | Yes | Byte select |
| $\overline{SIO2}$ | Yes | Byte select |
| $\overline{SIO3}$ | Yes | Byte select |
| Cn | No | Inactive |

Status Signals

ZERO = 1 if result (selected bytes) = 0
 N = 0
 OVR = 0
 C = 0

EXAMPLE (assumes a 32-bit configuration)

Logically AND bytes 1 and 2 of register 3 with input on the DB bus; store the result in register 3.

| Instr Code | Oprd Addr | Oprd Addr | Oprd Sel EB1- EA EB0 | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2- CF0 | SIO3- SIO0 | IESIO3- IESIO0 |
|---------------|--------------|--------------|----------------------------|-----------------------|---------------------|------|-------------------|-----|-----|------|------|------|----|-------------|---------------|-------------------|
| | | | | | SELMO | WE0 | SELRF1- SELRF0 | OE0 | OE1 | OEY0 | OEY1 | OEY2 | | | | |
| 1110 1000 | 00 0011 | XX XXXX | 0 10 | 00 0011 | 0 | 0000 | 10 | X | X | XXXX | 0 | X | X | 110 | 1001 | 0000 |

Assume register file 3 holds 398FBEBE (Hex) and input on the DB port is 4290BFBF (Hex):

Source 0011 1001 1000 1111 1011 1110 1011 1110 Rn ← RF(3)n

Source 0100 0010 1001 0000 1011 1111 1011 1111 Sn ← DBn

Destination 0100 0010 1000 0000 1011 1111 1011 1111 RF(3)n ← Fn or Sn[†]

[†]F = ALU result

n = nth byte

Register file 3 gets F if byte selected, S if byte not selected.

| Signal | Programmable | User |
|--------|--------------|-------------|
| SIO3 | Yes | Byte select |
| SIO2 | Yes | Byte select |
| SIO1 | Yes | Byte select |
| SIO0 | Yes | Byte select |
| Cn | No | Inactive |

FUNCTION

Converts a BCD number to binary.

DESCRIPTION

This instruction allows the user to convert an N-digit BCD number to a 4N-bit binary number in 4(N-1) plus 8 clocks. The instruction sums the R and S buses with carry.

A one-bit arithmetic left shift is performed on the ALU output. A zero is filled into bit 0 of the least significant byte unless $\overline{SIO0}$ is set low, which would force bit 0 to one. Bit 7 of the most significant byte is dropped.

Simultaneously, the contents of the MQ register are rotated one bit to the left. Bit 7 of the most significant byte is rotated to bit 0 of the least significant byte.

Recommended R Bus Source Operands

| | | | |
|---------------|----------------|---------|------------------------------|
| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
| Yes | No | No | No |

Recommended S Bus Source Operands

| | | |
|---------------|---------|----------------|
| RF (B5-B0) | DB-Port | MQ Register |
| Yes | Yes | No |

Recommended Destination Operands

| | | |
|---------------|---------------|--------|
| RF (C5-C0) | RF (B5-B0) | Y-Port |
| Yes | No | No |

Shift Operations

| | |
|------|------|
| ALU | MQ |
| Left | Left |

Control/Data Signals

| Signal | User Programmable | Use |
|--------------------------|-------------------|---|
| SSF | No | Inactive |
| $\overline{\text{SIO0}}$ | Yes | If high or floating, fills a zero in LSB of ALU shifter; if low, fills a one in LSB of ALU shifter. |
| $\overline{\text{SIO1}}$ | No | Inactive in 32-bit configuration. Used in other |
| $\overline{\text{SIO2}}$ | No | configurations to select endfill in LSBs. |
| $\overline{\text{SIO3}}$ | No | |
| Cn | Yes | Should be programmed low for proper conversion. |

Status Signals

| | |
|------|-----------------------------------|
| ZERO | = 1 if result = 0 |
| N | = 1 if MSB = 1 |
| OVR | = 1 if signed arithmetic overflow |
| C | = 1 if carry-out = 1 |

ALGORITHM

The following code converts an N-digit BCD number to a 4N-bit binary number in 4(N-1) plus 8 clocks. This is one possible user generated algorithm. It employs the standard conversion formula for a BCD number (shown here for 32 bits):

$$\text{ABCD} = [(A \times 10 + B) \times 10 + C] \times 10 + D.$$

The conversion begins with the most significant BCD digit. Addition is performed in radix 2.

PSEUDOCODE

| | | |
|----------------------------|-----------------------|---|
| LOADMQ | NUM | Load MQ with BCD number. |
| SUB | ACC, ACC, SLCMQ | Clear accumulator; Circular left shift MQ. |
| SUB | MSK, MSK, SLCMQ | Clear mask register; Circular left shift MQ. |
| SLCMQ | | Circular left shift MQ. |
| SLCMQ | | Circular left shift MQ. |
| ADDI | ACC, MSK, 15 | Store 15 in mask register. |
| Repeat N-1 times: | | |
| (N = number of BCD digits) | | |
| AND | MQ, MSK, R1, SLCMQ | Extract one digit; Circular left shift MQ. |
| ADD | ACC, R1, R1, SLCMQ | Add extracted digit to accumulator, and store result in R1; Circular left shift MQ. |
| BCDBIN | R1, R1, ACC | Perform BCDBIN instruction, and store result in accumulator [$4 \times (\text{ACC} + 4 \times \text{digit})$]; Circular left shift MQ. |
| BCDBIN | ACC, R1, ACC | Perform BCDBIN instruction, and store result in accumulator [$10 \times (\text{ACC} + 10 \times \text{digit})$]; Circular left shift MQ. |
| (END REPEAT) | | |
| AND | MQ MSK, R1 | Fetch last digit. |
| ADD | ACC, R1, ACC | Add in last digit and store result in accumulator. |

FUNCTION

$S' + C_n$ for selected bytes of S.

DESCRIPTION

Bytes with $\overline{SIO0}$ programmed low compute $S' + C_n$. Bytes with $\overline{SIO0}$ programmed high pass S unaltered. Multiple bytes can be selected only if they are adjacent to one another. At least one byte must be nonselected.

Available R Bus Source Operands

| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
|---------------|----------------|---------|------------------------------|
| No | No | No | No |

Available S Bus Source Operands

| RF (B5-B0) | DB-Port | MQ Register |
|---------------|---------|----------------|
| Yes | Yes | Yes |

Available Destination Operands Shift Operations

| RF (C5-C0) | RF (B5-B0) | Y-Port | ALU | MQ |
|---------------|---------------|--------|------|------|
| Yes | No | Yes | None | None |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|----------------------|---|
| SSF | No | Inactive |
| $\overline{SIO0}$ | Yes | Byte select |
| $\overline{SIO1}$ | Yes | Byte select |
| $\overline{SIO2}$ | Yes | Byte select |
| $\overline{SIO3}$ | Yes | Byte select |
| C_n | Yes | Propagates through nonselected bytes; increments selected byte(s) if programmed high. |

Status Signals

| | |
|------|---|
| ZERO | = 1 if result (selected bytes) = 0 |
| N | = 0 |
| OVR | = 1 if signed arithmetic overflow (selected bytes) |
| C | = 1 if carry-out (most significant selected byte) = 1 |

EXAMPLE (assumes a 32-bit configuration)

Invert bytes 0 and 1 of register 3 and add them to the carry (bytes 2 and 3 are not changed). Store the result in register 3.

| Instr Code | Oprd Addr | Oprd Addr | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2- CF0 | SIO3- SIO0 | IESIO3- IESIO0 |
|---------------|--------------|--------------|----------------------------|-----------------------|---------------------|-------------|-------------------|------------|------------|-------------|------------|---|-----|-------------|---------------|-------------------|
| | | | | | SELMQ | WE3- WE0 | SELRF1- SELRF0 | OEA OEA | OEB OEB | OY3- OY0 | OES OES | | | | | |
| 1100 1000 | XX XXXX | 00 0001 | X 00 | 00 0011 | 0 | 0000 | 10 | X | X | XXXX | 0 | 1 | 110 | 1100 | 0000 | |

Assume register file 3 holds A3018181 (Hex):

Source

1010 0011 0000 0001 1000 0001 1000 0001

Sn ← RF(3)n

ALU

0101 1100 1111 1110 0111 1110 0111 1111

F_n ← S'n + C_n

Destination

1010 0011 0000 0001 0111 1110 0111 1111

RF(3)n ← F_n or S_n[†]

[†]F = ALU result
n = nth byte
Register file 3 gets F if byte selected, S if byte not selected.

| Signal | Programmable | Use |
|--------|--------------|--|
| Cn | Yes | Propagates through nonselected bytes; increments selected byte(s) if programmed high |
| SIO3 | Yes | Byte select |
| SIO2 | Yes | Byte select |
| SIO1 | Yes | Byte select |
| SIO0 | Yes | Byte select |
| SIF | No | Inactive |

FUNCTION

Increments selected bytes of S if the carry is set.

DESCRIPTION

Bytes with $\overline{\text{SIO}}$ inputs programmed low compute $S + \text{Cn}$. Bytes with $\overline{\text{SIO}}$ inputs programmed high, pass S unaltered. Multiple bytes can be selected only if they are adjacent to one another. At least one byte must be nonselected.

Available R Bus Source Operands

| | | | |
|---------------|----------------|---------|------------------------------|
| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
| No | No | No | No |

Available S Bus Source Operands

| | | |
|---------------|---------|----------------|
| RF (B5-B0) | DB-Port | MQ Register |
| Yes | Yes | Yes |

Available Destination Operands Shift Operations

| | | | | |
|---------------|---------------|--------|------|------|
| RF (C5-C0) | RF (B5-B0) | Y-Port | ALU | MQ |
| Yes | No | Yes | None | None |

Control/Data Signals

| Signal | User Programmable | Use |
|--------------------------|----------------------|---|
| SSF | No | Inactive |
| $\overline{\text{SIO0}}$ | Yes | Byte select |
| $\overline{\text{SIO1}}$ | Yes | Byte select |
| $\overline{\text{SIO2}}$ | Yes | Byte select |
| $\overline{\text{SIO3}}$ | Yes | Byte select |
| Cn | Yes | Propagates through nonselected bytes; increments selected byte(s) if programmed high. |

Status Signals

ZERO = 1 if result (selected bytes) = 0
 N = 0
 OVR = 1 if signed arithmetic overflow (selected bytes)
 C = 1 if carry-out (most significant selected byte) = 1

EXAMPLE (assumes a 32-bit configuration)

Add bytes 1 and 2 of register 7 to the carry (bytes 0 and 3 are not changed). Store the result in register 2.

| Instr Code | Oprd Addr | Oprd Addr | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF0 | SIO3- SIO0 | IESIO3- IESIO0 |
|---------------|--------------|--------------|----------------------------|-----------------------|---------------------|-------------|-------------------|-------------|---------------|------|---|---|-----|------|---------------|-------------------|
| | | | | | SELMO | WE3- WE0 | SELRF1- SELRF0 | OE3- OE0 | OEY3- OEY0 | OES | | | | | | |
| 1011 1000 | XX XXXX | 00 0111 | X 00 | 00 0010 | 0 | 0000 | 10 | X | X | XXXX | 0 | 1 | 110 | 1100 | 0000 | |

Assume register file 7 holds 408FBEBE (Hex):

Source 0100 0000 1000 1111 1011 1110 1011 1110 $S_n \leftarrow RF(7)n$

ALU 0100 0000 1000 1111 1011 1111 1011 1110 $F_n \leftarrow S_n + C_n$

Destination 0100 0000 1000 1111 1011 1111 1011 1110 $RF(2)n \leftarrow F_n \text{ or } S_n^\dagger$

$^\dagger F$ = ALU result

n = nth byte

Register file 11 gets F if byte selected, S if byte not selected.

FUNCTION

Converts a binary number to excess-3 representation.

DESCRIPTION

This instruction converts an N-digit binary number to a N/4 digit excess-3 number representation in $2N + 3$ clocks. The data on the R and S buses are added to the carry-in, which contains the most significant bit of the MQ register. The contents of the MQ register are rotated one bit to the left. The most significant bit is shifted out and passed to the least significant bit position. Depending on the configuration selected, this shift may be within the same byte or from the most significant byte to the least significant byte.

Recommended R Bus Source Operands

| | | | |
|---------------|----------------|---------|------------------------------|
| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
| Yes | No | No | No |

Recommended S Bus Source Operands

| | | |
|---------------|---------|----------------|
| RF (B5-B0) | DB-Port | MQ Register |
| Yes | Yes | No |

Recommended Destination Operands

| | | |
|---------------|---------------|--------|
| RF (C5-C0) | RF (B5-B0) | Y-Port |
| Yes | No | Yes |

Shift Operations

| | |
|------|------|
| ALU | MQ |
| None | Left |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|-------------------|---------------------------|
| SSF | No | Inactive |
| $\overline{SIO0}$ | No | Inactive |
| $\overline{SIO1}$ | No | Inactive |
| $\overline{SIO2}$ | No | Inactive |
| $\overline{SIO3}$ | No | Inactive |
| Cn | No | Holds MSB of MQ register. |

Status Signals

ZERO = 1 if result = 0

N = 1 if MSB = 1

OVR = 1 if signed arithmetic overflow

C = 1 if carry-out = 1

ALGORITHM

The following code converts an N-digit binary number to a N/4 digit excess-3 number in $2N+3$ clocks. It employs the standard conversion formula for a binary number:

$$a_n 2^n + a_{n-1} 2^{n-1} + a_{n-2} 2^{n-2} + \dots + a_0 = \{[(2a_n + a_{n-1}) \times 2 + a_{n-1}] \times 2 + \dots + a_0\} \times 2 + a_0.$$

The conversion begins with the most significant bit. Addition during the BINEX3 instruction is performed in radix 10 (excess-3).

LOADMQ NUM

Load MQ with binary number.

SUB ACC, ACC, ACC

Clear accumulator;

SET1 ACC, 33 (Hex)

Store 33 (Hex) in all bytes of accumulator.

Repeat N times:

(N = number of bits in binary number)

BINEX3 ACC, ACC, ACC

Double accumulator and add in most significant bit of MQ register. Circular left shift MQ.

EX3C ACC

Perform excess-3 correction.

(END REPEAT)

| Signal | Programmable | Use |
|--------|--------------|-------------|
| Cn | No | Inactive |
| St03 | Yes | Byte select |
| St02 | Yes | Byte select |
| St01 | Yes | Byte select |
| St00 | Yes | Byte select |
| Stf | No | Inactive |

FUNCTION

Evaluates R OR S of selected bytes.

DESCRIPTION

Bytes with \overline{SIO} inputs programmed low evaluate R OR S. Bytes with \overline{SIO} inputs programmed high, pass S unaltered. Multiple bytes can be selected only if they are adjacent to one another. At least one byte must be nonselected.

Available R Bus Source Operands

| | | | |
|---------------|----------------|---------|-----------------------------|
| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 : A3-A0 Mask |
| Yes | No | Yes | No |

Available S Bus Source Operands

| | | |
|---------------|---------|----------------|
| RF (B5-B0) | DB-Port | MQ Register |
| Yes | Yes | Yes |

Available Destination Operands Shift Operations

| | | | | |
|---------------|---------------|--------|------|------|
| RF (C5-C0) | RF (B5-B0) | Y-Port | ALU | MQ |
| Yes | No | Yes | None | None |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|----------------------|-------------|
| SSF | No | Inactive |
| $\overline{SIO}0$ | Yes | Byte select |
| $\overline{SIO}1$ | Yes | Byte select |
| $\overline{SIO}2$ | Yes | Byte select |
| $\overline{SIO}3$ | Yes | Byte select |
| Cn | No | Inactive |

Status Signals

ZERO = 1 if result (selected bytes) = 0

N = 0

OVR = 0

C = 0

EXAMPLE (assumes a 32-bit configuration)

Logically OR bytes 1 and 2 of register 12 with bytes 1 and 2 on the DB bus. Concatenate with DB bytes 0 and 3, storing the result in register 12.

| Instr Code | Oprd Addr | Oprd Addr | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2- CF0 | SIO3- SIO0 | IESIO3- IESIO0 |
|---------------|--------------|--------------|----------------------------|-----------------------|---------------------|----------------|----------------|-----|---------------|------|---|---|----|-------------|---------------|-------------------|
| | | | | | WE3- SELMO | SELRF1- WEO | SELRF0- OEA | OEB | OEY3- OEY0 | OES | | | | | | |
| 1111 1000 | 00 1100 | XX XXXX | 0 10 | 00 1100 | 0 | 0000 | 10 | X | X | XXXX | 0 | X | X | 110 | 1001 | 0000 |

Assume register file 12 holds 578FBEBE (Hex) and the DB bus holds 1C90BEBE (Hex):

Source

0101 0111 1000 1111 1011 1110 1011 1110

Rn ← RF(12)n

Source

0001 1100 1001 0000 1011 1110 1011 1100

Sn ← DBn

Destination

0001 1100 1001 1111 1011 1110 1011 1110

RF(12)n ← Fn or Sn[†]

†F = ALU result

n = nth package

Register file 12 gets F if byte selected, S if byte not selected.

| Signal | Programmable | Use |
|--------|--------------|--|
| STP | No | inactive |
| SIO0 | Yes | Byte select |
| SIO1 | Yes | Byte select |
| SIO2 | Yes | Byte select |
| SIO3 | Yes | Byte select |
| Cn | Yes | Propagates through nonselected bytes; should be set high for two's complement subtraction. |

FUNCTION

Subtracts R from S in selected bytes.

DESCRIPTION

Bytes with \overline{SIO} inputs programmed low compute $R' + S + Cn$. Bytes with \overline{SIO} inputs programmed high, pass S unaltered. Multiple bytes can be selected only if they are adjacent to one another. At least one byte must be nonselected.

Available R Bus Source Operands

| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
|---------------|----------------|---------|------------------------------|
| Yes | No | Yes | No |

Available S Bus Source Operands

| RF (B5-B0) | DB-Port | MQ Register |
|---------------|---------|----------------|
| Yes | Yes | Yes |

Available Destination Operands Shift Operations

| RF (C5-C0) | RF (B5-B0) | Y-Port | ALU | MQ |
|---------------|---------------|--------|------|------|
| Yes | No | Yes | None | None |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|----------------------|--|
| SSF | No | Inactive |
| $\overline{SIO0}$ | Yes | Byte select |
| $\overline{SIO1}$ | Yes | Byte select |
| $\overline{SIO2}$ | Yes | Byte select |
| $\overline{SIO3}$ | Yes | Byte select |
| Cn | Yes | Propagates through nonselected bytes; should be set high for two's complement subtraction. |

Status Signals

ZERO = 1 if result (selected bytes) = 0
 N = 0
 OVR = 1 if signed arithmetic overflow (selected bytes)
 C = 1 if carry-out (most significant selected byte) = 1

EXAMPLE (assumes a 32-bit configuration)

Subtract bytes 1 and 2 of register 1 with carry from bytes 1 and 2 of register 3. Concatenate with bytes 0 and 3 of register 3, storing the result in register 11.

| Instr Code | Oprd Addr | Oprd Addr | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2- CF0 | SIO3- SIO0 | IESIO3- IESIO0 |
|---------------|--------------|--------------|----------------------------|-----------------------|---------------------|------|--------|-----|-----|------|-----|---|-----|-------------|---------------|-------------------|
| | | | | | SELMO | WE0 | SELRFO | OEA | OEB | OEY0 | OES | | | | | |
| 1010 1000 | 00 0001 | 00 0011 | 0 00 | 00 1011 | 0 | 0000 | 10 | X | X | XXXX | 0 | 1 | 110 | 1001 | 0000 | |

Assume register file 1 holds 091B5858 (Hex) and register file 3 holds 703A9898 (Hex):

Source 0000 1001 0001 1011 0101 1000 0101 1000 $R_n \leftarrow RF(1)n$

Source 0111 0000 0011 1010 1001 1000 1001 1000 $S_n \leftarrow RF(3)n$

ALU 0110 0111 0001 1111 0100 0000 0100 0000 $F_n \leftarrow R'n + S_n + C_n$

Destination 0111 0000 0001 1111 0100 0000 1001 1000 $RF(11)n \leftarrow F_n \text{ or } S_n^\dagger$

$^\dagger F$ = ALU result

n = nth package

Register file 11 gets F if byte selected, S if byte not selected.

| Signal | Programmable | User |
|---|--------------|-------------|
| set high for two's complement subtraction | Yes | |
| Propagates through nonselected bytes; should be | Yes | |
| Cn | Yes | |
| SIO3 | Yes | Byte select |
| SIO2 | Yes | Byte select |
| SIO1 | Yes | Byte select |
| SIO0 | Yes | Byte select |
| set | No | inactive |

FUNCTION

Subtracts S from R in selected bytes.

DESCRIPTION

Bytes with \overline{SIO} inputs programmed low compute $R + S' + Cn$. Bytes with \overline{SIO} inputs programmed high, pass S unaltered. Multiple bytes can be selected only if they are adjacent to one another. At least one byte must be nonselected.

Available R Bus Source Operands

| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
|---------------|----------------|---------|------------------------------|
| Yes | No | Yes | No |

Available S Bus Source Operands

| RF (B5-B0) | DB-Port | MQ Register |
|---------------|---------|----------------|
| Yes | Yes | Yes |

Available Destination Operands Shift Operations

| RF (C5-C0) | RF (B5-B0) | Y-Port |
|---------------|---------------|--------|
| Yes | No | Yes |

| ALU | MQ |
|------|------|
| None | None |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|-------------------|--|
| SSF | No | Inactive |
| $\overline{SIO0}$ | Yes | Byte select |
| $\overline{SIO1}$ | Yes | Byte select |
| $\overline{SIO2}$ | Yes | Byte select |
| $\overline{SIO3}$ | Yes | Byte select |
| Cn | Yes | Propagates through nonselected bytes; should be set high for two's complement subtraction. |

Status Signals

| | |
|------|---|
| ZERO | = 1 if result (selected bytes) = 0 |
| N | = 0 |
| OVR | = 1 if signed arithmetic overflow (selected bytes) |
| C | = 1 if carry-out (most significant selected byte) = 1 |

EXAMPLE (assumes a 32-bit configuration)

Subtract bytes 1 and 2 of register 3 with carry from bytes 1 and 2 of register 1. Concatenate with bytes 0 and 3 of register 3, storing the result in register 11.

| Instr Code | Oprd Addr | Oprd Addr | Oprd Sel EB1- EA EB0 | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2- CF0 | SIO3- SIO0 | IESIO3- IESIO0 |
|---------------|--------------|--------------|----------------------------|-----------------------|---------------------|------|--------|-----|-----|------|-----|--|----|-------------|---------------|-------------------|
| | | | | | SELMO | WE0 | SELRF0 | OEA | OEB | OEY0 | OES | | | | | |
| 1001 1000 | 00 0001 | 00 0011 | 0 00 | 00 1011 | 0 | 0000 | 10 | X | X | XXXX | 0 | | 1 | 110 | 1001 | 0000 |

Assume register file 1 holds 5288B8B8 (Hex) and register file 3 holds 143A9898 (Hex):

Source

0101 0010 1000 1000 1011 1000 1011 1000

Rn ← RF(1)n

Source

0001 0100 0011 1010 1001 1000 1001 1000

Sn ← RF(3)n

ALU

0011 1110 0100 1110 0010 0000 0010 0000

Fn ← Rn + S'n + Cn

Destination

0101 0010 0100 1110 0010 0000 1011 1000

RF(11)n ← Fn or Sn[†]

†F = ALU result

n = nth byte

Register file 11 gets F if byte selected, S if byte not selected.

| Signal | Programmable | Use |
|--------|--------------|-------------|
| Cn | No | Inactive |
| SIO3 | Yes | Byte select |
| SIO2 | Yes | Byte select |
| SIO1 | Yes | Byte select |
| SIO0 | Yes | Byte select |
| SSP | No | Inactive |

Byte XOR R and S **BXOR** (Byte Exclusive OR R and S)

D 8

FUNCTION

Evaluates R exclusive OR S in selected bytes.

DESCRIPTION

Bytes with \overline{SIO} inputs programmed low evaluate R exclusive OR S. Bytes with \overline{SIO} inputs programmed high, pass S unaltered. Multiple bytes can be selected only if they are adjacent to one another. At least one byte must be nonselected.

Available R Bus Source Operands

| | | | |
|---------------|----------------|---------|-----------------------------|
| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 : A3-A0 Mask |
| Yes | No | Yes | No |

Available S Bus Source Operands

| | | |
|---------------|---------|----------------|
| RF (B5-B0) | DB-Port | MQ Register |
| Yes | Yes | Yes |

Available Destination Operands Shift Operations

| | | | | |
|---------------|---------------|--------|------|------|
| RF (C5-C0) | RF (B5-B0) | Y-Port | ALU | MQ |
| Yes | No | Yes | None | None |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|-------------------|-------------|
| SSF | No | Inactive |
| $\overline{SIO0}$ | Yes | Byte select |
| $\overline{SIO1}$ | Yes | Byte select |
| $\overline{SIO2}$ | Yes | Byte select |
| $\overline{SIO3}$ | Yes | Byte select |
| Cn | No | Inactive |

Status Signals

| | |
|------|------------------------------------|
| ZERO | = 1 if result (selected bytes) = 0 |
| N | = 0 |
| OVR | = 0 |
| C | = 0 |

EXAMPLE (assumes a 32-bit configuration)

Exclusive OR bytes 1 and 2 of register 6 with bytes 1 and 2 on the DB bus; concatenate the result with DB bytes 0 and 3, storing the result in register 10.

| Instr Code | Oprd Addr | Oprd Addr | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2- CF0 | SIO3- SIO0 | IESIO3- IESIO0 |
|---------------|--------------|--------------|----------------------------|-----------------------|---------------------|----------------|--------|-------------|-------------|-------------|-------------|-------------|-----|-------------|---------------|-------------------|
| | | | | | WE3- SELMO | SELRF1- WE0 | SELRF0 | OE3- OE0 | OE2- OE1 | OE1- OE0 | OE0- OE0 | OE0- OE0 | | | | |
| 1101 1000 | 00 0110 | XX XXXX | 0 10 | 00 1010 | 0 | 0000 | 10 | X | X | XXXX | 0 | 1 | 110 | 1001 | 0000 | |

Assume register file 6 holds 938FBEBE (Hex) and the DB bus holds 4190BEBE (Hex):

| | | |
|--------|---|-------------|
| Source | 1001 0011 1000 1111 1011 1110 1011 1110 | Rn ← RF(6)n |
| Source | 0100 0001 1001 0000 1011 1110 1011 1110 | Sn ← DBn |

| | | |
|-------------|---|---------------------------------|
| Destination | 0100 0001 0001 1111 0000 0000 1011 1110 | RF(10)n ← Fn or Sn [†] |
|-------------|---|---------------------------------|

[†]F = ALU result
n = nth package
Register file 10 gets F if byte selected, S if byte not selected.

| | | |
|---|---|---|
| 1 | F | † |
|---|---|---|

CLEAR**CLR****FUNCTION**

Forces ALU output to zero and clears the BCD flip-flops.

DESCRIPTION

ALU output is forced to zero and the BCD flip-flops are cleared.

† This instruction may also be coded with the following opcodes:

[2] [F], [3] [F], [4] [F], [6] [F], [B] [F], [C] [F], [E] [F]

Available R Bus Source Operands

| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
|---------------|----------------|---------|------------------------------|
| No | No | No | No |

Available S Bus Source Operands

| RF (B5-B0) | DB-Port | MQ Register |
|---------------|---------|----------------|
| No | No | No |

Available Destination Operands

| RF (C5-C0) | RF (B5-B0) | Y-Port |
|---------------|---------------|--------|
| Yes | No | Yes |

Shift Operations

| ALU | MQ |
|------|------|
| None | None |

Status Signals

| |
|----------|
| ZERO = 1 |
| N = 0 |
| OVR = 0 |
| Cn = 0 |

FUNCTION

Evaluates R exclusive OR S for use with cyclic redundancy check codes.

DESCRIPTION

Data on the R bus is exclusive ORed with data on the S bus. If MQ0 XNORed with S0 is zero (MQ0 is the LSB of the MQ register and S0 is the LSB of S-bus data), the result is sent to the ALU shifter. Otherwise, data on the S bus is sent to the ALU shifter.

A right shift is performed; the MSB is filled with R0 (MQ0 XOR S0), where R0 is the LSB of R-bus data. A circular right shift is performed on MQ data.

Recommended R Bus Source Operands

| | | | |
|---------------|----------------|---------|-----------------------------|
| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 : A3-A0 Mask |
| Yes | No | No | No |

Recommended S Bus Source Operands

| | | |
|---------------|---------|----------------|
| RF (B5-B0) | DB-Port | MQ Register |
| Yes | Yes | No |

Recommended Destination Operands

| | | |
|---------------|---------------|--------|
| RF (C5-C0) | RF (B5-B0) | Y-Port |
| Yes | No | No |

Shift Operations

| | |
|-------|-------|
| ALU | MQ |
| Right | Right |

Control/Data Signals

| Signal | User Programmable | Use |
|--------------------------|-------------------|----------|
| SSF | No | Inactive |
| $\overline{\text{SIO0}}$ | No | Inactive |
| $\overline{\text{SIO1}}$ | No | Inactive |
| $\overline{\text{SIO2}}$ | No | Inactive |
| $\overline{\text{SIO3}}$ | No | Inactive |
| Cn | No | Inactive |

Status Signals

| | | |
|------|---|-----------------|
| ZERO | = | 1 if result = 0 |
| N | = | 0 |
| OVR | = | 0 |
| Cn | = | 0 |

CYCLIC REDUNDANCY CHARACTER CHECK

DESCRIPTION

Serial binary data transmitted over a channel is susceptible to error bursts. These bursts may be detected and corrected by standard encoding methods such as cyclic redundancy check codes, fire codes, or computer generated codes. These codes all divide the message vector by a generator polynomial to produce a remainder that contains parity information about the message vector.

If a message vector of m bits, $a(x)$, is divided by a generator polynomial, $g(x)$, of order $k-1$, a k bit remainder, $r(x)$, is formed. The code vector, $c(x)$, consisting of $m(x)$ and $r(x)$ of length $n = m + k$ is transmitted down the channel. The receiver divides the received vector by $g(x)$.

After m divide iterations, $r(x)$ will be regenerated only if there is no error in the message bits. After k more iterations, the result will be zero if and only if no error has occurred in either the message or the remainder.

ALGORITHM

An algorithm for a cyclic redundancy character check, using the 'ACT8832A as a receiver, is given below:

| | |
|----------------------|--|
| LOADMQ VEC(X) | Load MQ with first 32 message bits of received vector $c'(x)$. |
| LOAD POLY | Load register with polynomial $g(x)$. |
| CLEAR SUM | Clear register acting as accumulator. |
| REPEAT (n/32) TIMES: | |
| SUM = SUM CRC POLY | Perform CRC instruction where R Bus = POLY S Bus = SUM Store result in SUM. |
| LOADMQ VEC(X) | Load MQ with next 32 message bits of received vector $c'(x)$. |
| (END REPEAT) | |

SUM now contains the remainder $[r'(x)]$ of $c'(x)$. A syndrome generation routine may be called next, if required.

Note that the most significant bit of

$$g(x) = (g_{k-1})(x^{k-1}) + (g_{k-2})(x^{k-2}) + \dots (g_0)(x^0)$$

is implied and that POLY(0) is set to zero if the length of $g(x)$ requires fewer bits than are in the machine word width.

FUNCTION

Corrects the remainder of nonrestoring division routine if correction is required.

DESCRIPTION

DIVRF tests the result of the final step in nonrestoring division iteration: SDIVIT (for signed division) or UDIVIT (for unsigned division). An error in the remainder results when it is nonzero and the signs of the remainder and the dividend are different.

The R bus must be loaded with the divisor and the S bus with the most significant half of the previous result. The least significant half is in the MQ register. The Y bus result must be stored in the register file for use during the subsequent SDIVQF instruction.

DIVRF tests to determine whether a fix is required and evaluates:

$Y \leftarrow S + R' + 1$ if a fix is necessary

$Y \leftarrow S + R + 0$ if a fix is unnecessary

Overflow is reported to OVR at the end of the division routine (after SDIVQF).

Recommended R Bus Source Operands

| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
|---------------|----------------|---------|------------------------------|
| Yes | No | No | No |

Recommended S Bus Source Operands

| RF (B5-B0) | DB-Port | MQ Register |
|---------------|---------|----------------|
| Yes | Yes | No |

Recommended Destination Operands

| RF (C5-C0) | RF (B5-B0) | Y-Port |
|---------------|---------------|--------|
| Yes | No | No |

Shift Operations

| ALU | MQ |
|------|------|
| None | None |

Control/Data Signals

| Signal | User Programmable | Use |
|--------------------------|-------------------|---------------------------|
| SSF | No | Inactive |
| $\overline{\text{SIO0}}$ | No | Inactive |
| $\overline{\text{SIO1}}$ | No | Inactive |
| $\overline{\text{SIO2}}$ | No | Inactive |
| $\overline{\text{SIO3}}$ | No | Inactive |
| Cn | Yes | Should be programmed high |

Status Signals

| | |
|------|----------------------|
| ZERO | = 1 if remainder = 0 |
| N | = 0 |
| OVR | = 0 |
| Cn | = 1 if carry-out = 1 |

FUNCTION

Tests the two most significant bits of a double precision number. If they are the same, shifts the number to the left.

DESCRIPTION

This instruction is used to normalize a two's complement, double precision number by shifting the number one bit to the left and filling a zero into the LSB unless $\overline{SIO0}$ is low. The S bus holds the most significant half; the MQ register holds the least significant half.

Normalization is complete when overflow occurs. The shift is inhibited whenever normalization is attempted on a number already normalized.

Available R Bus Source Operands

| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
|---------------|----------------|---------|------------------------------|
| No | No | No | No |

Recommended S Bus Source Operands (MSH)

| RF (B5-B0) | DB-Port | MQ Register |
|---------------|---------|----------------|
| Yes | No | No |

Recommended Destination Operands

| RF (C5-C0) | RF (B5-B0) | Y-Port |
|---------------|---------------|--------|
| Yes | No | No |

**Shift Operations
(conditional)**

| ALU | MQ |
|------|------|
| Left | Left |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|-------------------|---|
| SSF | No | Inactive |
| $\overline{SIO0}$ | Yes | When low, selects a one end-fill bit in LSB |
| $\overline{SIO1}$ | No | Passes internally generated end-fill bits |
| $\overline{SIO2}$ | No | |
| $\overline{SIO3}$ | No | |
| Cn | No | |

Status Signals

ZERO = 1 if result = 0
 N = 1 if MSB = 1
 OVR = 1 if MSB XOR 2nd MSB = 1
 Cn = 0

EXAMPLE (assumes a 32-bit configuration)

Normalize a double-precision number.

(This example assumes that the MSH of the number to be normalized is in register 3 and the LSH is in the MQ register. The zero on the OVR pin at the end of the instruction cycle indicates that normalization is not complete and the instruction should be repeated).

| Instr Code 17-10 | Oprd Addr A5-A0 | Oprd Addr B5-B0 | Oprd Sel EB1- EA EB0 | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2 CF0 |
|------------------------|-----------------------|-----------------------|----------------------------|-----------------------|---------------------|----------------|--------|--------------|--------------|------|---|---|-----|------------|
| | | | | | WE3- SELMQ | SELRF1- WE0 | SELRFO | OE3- OEY0 | OEY3- OES | | | | | |
| 0011 0000 | XX XXXX | 00 0011 | X 00 | 00 0011 | 0 | 0000 | 10 | X | X | XXXX | 0 | X | 110 | |

Assume register file 3 holds FA75D84E (Hex) and MQ register holds 37F6D843 (Hex):

Source 1111 1010 0111 0101 1101 1000 0100 1110 ALU shifter \leftarrow RF(3)

Source 0011 0111 1111 0110 1101 1000 0100 0011 MQ shifter \leftarrow MQ register

Destination 1111 0100 1110 1011 1011 0000 1001 1101 8RF(3) \leftarrow Result (MSH)

Destination 0110 1111 1110 1101 1011 0000 1000 0110 MQ register \leftarrow Result (LSH)

0 OVR \leftarrow 0[†]

[†]Normalization not complete at the end of this instruction cycle.

FUNCTION

Output contents of the divide/BCD flip-flops.

DESCRIPTION

The contents of the divide/BCD flip-flops are passed through the MQ register to the Y output Imultiplexer.

Available R Bus Source Operands

| | | | |
|---------|-------|---------|-------|
| RF | A3-A0 | DA-Port | C3-C0 |
| (A5-A0) | Immed | | :: |
| | | | A3-A0 |
| | | | Mask |
| No | No | No | No |

Available S Bus Source Operands

| | | |
|---------|---------|----------|
| RF | DB-Port | MQ |
| (B5-B0) | | Register |
| No | No | No |

Available Destination Operands Shift Operations

| | | | | |
|---------|---------|--------|------|------|
| RF | RF | Y-Port | ALU | MQ |
| (C5-C0) | (B5-B0) | | | |
| No | No | Yes | None | None |

Status Signals

| | | |
|------|---|---|
| ZERO | = | 0 |
| N | = | 0 |
| OVR | = | 0 |
| Cn | = | 0 |

EXAMPLES (assumes a 32-bit configuration)

Dump divide/BCD flip-flops to Y output.

| Instr Code I7-I0 | Oprd Addr A5-A0 | Oprd Addr B5-B0 | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2- CFO |
|------------------------|-----------------------|-----------------------|----------------------------|-----------------------|---------------------|-------------|-------------------|-------------|---------------|------|---|---|-----|-------------|
| | | | | | SELMQ | WE3- WE0 | SELRF1- SELRF0 | OE3- OE0 | OEY3- OEY0 | OES | | | | |
| 0101 1111 | XX XXXX | XX XXXX | X XX | XX XXXX | 1 | XXXX | XX | X | X | 0000 | X | X | 110 | |

Assume divide/BCD flip-flops contain 2A055470 (Hex):

Source 0010 1010 0000 0101 0101 0100 0111 0000 MQ register ← Divide/BCD flip-flops

Destination 0010 1010 0000 0101 0101 0100 0111 0000 Y output ← MQ register

FUNCTION

Corrects the result of excess-3 addition or subtraction in selected bytes.

DESCRIPTION

This instruction corrects excess-3 additions or subtractions in the byte mode. For correct excess-3 arithmetic, this instruction must follow each add or subtract. The operand must be on the S bus.

Data on the S bus is added to a constant on the R bus determined by the state of the BCD flip flops and previous overflow condition reported on the SSF pin. Bytes with $\overline{\text{SIO}}$ inputs programmed low evaluate the correct excess-3 representation. Bytes with $\overline{\text{SIO}}$ inputs programmed high or floating, pass S unaltered.

Available R Bus Source Operands

| | | | |
|---------------|----------------|---------|------------------------------|
| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
| No | No | No | No |

Available S Bus Source Operands

| | | |
|---------------|---------|----------------|
| RF (B5-B0) | DB-Port | MQ Register |
| Yes | No | No |

Available Destination Operands Shift Operations

| | | | | |
|---------------|---------------|--------|------------|----|
| RF (C5-C0) | RF (B5-B0) | Y-Port | \neg ALU | MQ |
| Yes | No | No | No | No |

Control/Data Signals

| Signal | User Programmable | Use |
|--------------------------|-------------------|-------------|
| SSF | No | Inactive |
| $\overline{\text{SIO0}}$ | Yes | Byte select |
| $\overline{\text{SIO1}}$ | Yes | Byte select |
| $\overline{\text{SIO2}}$ | Yes | Byte select |
| $\overline{\text{SIO3}}$ | Yes | Byte select |
| Cn | No | Inactive |

Status Signals

ZERO = 0

N = 0

OVR = 1 if arithmetic signed overflow

Cn = 1 if carry-out = 1

EXAMPLE (assumes a 32-bit configuration)

Add two BCD numbers and store the sum in register 3. Assume data comes in on DB bus.

1. Clear accumulator (SUB ACC, ACC)
2. Store 33 (Hex) in all bytes of register (SET1 R2, H/33/)
3. Add 33 (Hex) to selected bytes of first BCD number (BADD DB, R2, R1)
4. Add 33 (Hex) to selected bytes of second BCD number (BADD DB, R2, R3)
5. Add selected bytes of registers 1 and 3 (BADD, R1, R3, R3)
6. Correct the result (EX3BC, R3, R3)

| Instr Code I7-I0 | Oprd Addr A5-A0 | Oprd Addr B5-B0 | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2- CF0 | SIO3- SIO0 | IESIO3- IESIO0 |
|------------------------|-----------------------|-----------------------|----------------------------|-----------------------|---------------------|-------------|-------------------|-------------|---------------|------|---|---|-----|-------------|---------------|-------------------|
| | | | | | SELMO | WE3- WE0 | SELRF1- SELRF0 | OE3- OE0 | OEY3- OEY0 | OES | | | | | | |
| 1111 0010 | 00 0010 | XX XXXX | 0 XX | 00 0010 | 0 | 0000 | 10 | X | X | XXXX | 0 | 1 | 110 | XXXX | XXXX | |
| 0000 1000 | 00 0010 | XX XXXX | 0 XX | 00 0010 | 0 | 0000 | 10 | X | X | XXXX | 0 | X | 110 | XXXX | XXXX | |
| 1000 1000 | 00 0010 | XX XXXX | 0 10 | 00 0001 | 0 | 0000 | 10 | X | X | XXXX | 0 | 0 | 110 | 1100 | 0000 | |
| 1000 1000 | 00 0010 | XX XXXX | 0 10 | 00 0011 | 0 | 0000 | 10 | X | X | XXXX | 0 | 0 | 110 | 1100 | 0000 | |
| 1000 1000 | 00 0001 | 00 0011 | 0 00 | 00 0011 | 0 | 0000 | 10 | X | X | XXXX | 0 | 0 | 110 | 1100 | 0000 | |
| 1000 1111 | XX XXXX | 00 0011 | X 00 | 00 0011 | 0 | 0000 | 10 | X | X | XXXX | 0 | 0 | 110 | 1100 | 0000 | |

Assume DB bus holds 51336912 at third instruction and 34867162 at fourth instruction.

1 0000 0000 0000 0000 0000 0000 0000 0000 RF(2) ← 0

2 0000 0000 0000 0000 0011 0011 0011 0011 RF(2) ← 00003333 (Hex)

3 0101 0001 0011 0011 1001 1100 0100 0101 RF(1) ← RF(2) + DB

4 0011 0100 1000 0110 1010 0100 1001 0101 RF(3) ← RF(2) + DB

5 0011 0100 1000 0110 0100 0000 1101 1010 RF(3)n ← RF(1)n + RF(3)n

6 0011 0100 1000 0110 0100 0000 0111 0100 RF(3)n ← Corrected RF(3)n result

FUNCTION

Corrects the result of excess-3 addition or subtraction.

DESCRIPTION

This instruction corrects excess-3 additions or subtractions in the word mode. For correct excess-3 arithmetic, this instruction must follow each add or subtract. The operand must be on the S bus.

Data on the S bus is added to a constant on the R bus determined by the state of the BCD flip-flops and previous overflow condition reported by the SSF pin.

Available R Bus Source Operands

| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
|---------------|----------------|---------|------------------------------|
| No | No | No | No |

Available S Bus Source Operands

| RF (B5-B0) | DB-Port | MQ Register |
|---------------|---------|----------------|
| Yes | No | No |

Available Destination Operands Shift Operations

| RF (C5-C0) | RF (B5-B0) | Y-Port | ALU | MQ |
|---------------|---------------|--------|-----|----|
| Yes | No | Yes | No | No |

Control/Data Signals

| Signal | User Programmable | Use |
|--|----------------------|----------|
| SSF | No | Inactive |
| $\overline{\text{SI}}\overline{\text{O}}0$ | No | Inactive |
| $\overline{\text{SI}}\overline{\text{O}}1$ | No | Inactive |
| $\overline{\text{SI}}\overline{\text{O}}2$ | No | Inactive |
| $\overline{\text{SI}}\overline{\text{O}}3$ | No | Inactive |
| Cn | No | Inactive |

Status Signals

| | |
|------|-----------------------------------|
| ZERO | = 0 |
| N | = 1 if MSB = 1 |
| OVR | = 1 if arithmetic signed overflow |
| Cn | = 1 if carry-out = 1 |

EXAMPLE (assumes a 32-bit configuration)

Add two BCD numbers and store the sum in register 3. Assume data comes in on DA bus.

1. Clear accumulator (SUB ACC, ACC)
2. Store 33 (Hex) in all bytes of register (SET1 R2, H/33/)
3. Add 33 (Hex) to all bytes of first BCD number (ADD DB, R2, R1)
4. Add 33 (Hex) to all bytes of second BCD number (ADD DB, R2, R3)
5. Add the excess-3 data (ADD, R1, R3, R3)
6. Correct the excess-3 result (EX3C, R3, R3)
7. Subtract the excess-3 bias to go to BCD result.

| Instr Code I7-I0 | Oprd Addr A5-A0 | Oprd Addr B5-B0 | Oprd Sel EB1- EA EB0 | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2 CF0 |
|------------------------|-----------------------|-----------------------|----------------------------|-----------------------|---------------------|-------------|-------------------|--------------|--------------|---------------|--------------|---|-----|------------|
| | | | | | SELMO | WE3- WEO | SELRF1- SELRF0 | OEY3- OEA | OEY3- OEB | OEY3- OEYO | OEY3- OES | | | |
| 1111 0010 | 00 0010 | XX XXXX | 0 XX | 00 0010 | 0 | 0000 | 10 | X | X | XXXX | 0 | 1 | 110 | |
| 0000 1000 | 00 0010 | XX XXXX | 0 XX | 00 0010 | 0 | 0000 | 10 | X | X | XXXX | 0 | X | 110 | |
| 1111 0001 | 00 0010 | XX XXXX | 0 10 | 00 0001 | 0 | 0000 | 10 | X | X | XXXX | 0 | 0 | 110 | |
| 1111 0001 | 00 0010 | XX XXXX | 0 10 | 00 0011 | 0 | 0000 | 10 | X | X | XXXX | 0 | 0 | 110 | |
| 1111 0001 | 00 0001 | 00 0011 | 0 00 | 00 0011 | 0 | 0000 | 10 | X | X | XXXX | 0 | 0 | 110 | |
| 1001 1111 | XX XXXX | 00 0011 | X 00 | 00 0011 | 0 | 0000 | 10 | X | X | XXXX | 0 | 0 | 110 | |
| 1111 0010 | 00 0010 | 00 0011 | 0 00 | 00 0011 | 0 | 0000 | 10 | X | X | XXXX | 0 | 0 | 110 | |

Results of Instruction Cycles:

$$RF(2) \leftarrow 0$$

RF(2) ← 33333333 (Hex)

$$RF(1) \leftarrow RF(2) + DB$$
$$RF(3) \leftarrow RF(2) + DB$$
$$RF(3) \leftarrow RF(1) + RF(3)$$

RF(3) ← Corrected RF(3) result

$$RF(3) \leftarrow RF(3) - RF(2)$$

FUNCTION

Evaluates $R' + Cn$.

DESCRIPTION

Data on the R bus is inverted and added with carry. The result appears at the ALU and MQ shifters.

*The result of this instruction can be shifted in the same microcycle by specifying a shift instruction in the upper nibble (I7-I4) of the instruction field. The result may also be passed without shift. Possible instructions are listed in Table 15.

Available R Bus Source Operands

| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
|---------------|----------------|---------|------------------------------|
| Yes | No | Yes | No |

Available S Bus Source Operands

| RF (B5-B0) | DB-Port | MQ Register |
|---------------|---------|----------------|
| No | No | No |

Available Destination Operands

| RF (C5-C0) | RF (B5-B0) | Y-Port | ALU Shifter | MQ Shifter |
|---------------|---------------|--------|----------------|---------------|
| Yes | No | Yes | Yes | Yes |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|----------------------|--|
| SSF | No | Affect shift instructions programmed in bits I7-I4 of instruction field. |
| $\overline{STO0}$ | No | |
| $\overline{STO1}$ | No | |
| $\overline{STO2}$ | No | |
| $\overline{STO3}$ | No | |
| Cn | Yes | Increments if programmed high. |

Status Signals†

| | |
|------|-----------------------------------|
| ZERO | = 1 if result = 0 |
| N | = 1 if MSB = 1 |
| OVR | = 1 if signed arithmetic overflow |
| C | = 1 if carry-out = 1 |

†C is ALU carry out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Convert the data on the DA bus to two's complement and store the result in register 4.

| Instr Code | Oprd Addr A5-A0 | Oprd Addr B5-B0 | Oprd Sel EB1- EA EB0 | Dest Addr C5-C0 | Destination Selects | | | | | | | Cn | CF2- CFO |
|---------------|-----------------------|-----------------------|----------------------------|-----------------------|---------------------|-------------|-------------------|-------------|---------------|-------------|---|----|-------------|
| | | | | | SELMO | WE3- WE0 | SELRF1- SELRF0 | OE3- OE0 | OEY3- OEY0 | OE5- OE0 | | | |
| 1111 0111 | XX XXXX | XX XXXX | 1 XX | 00 0100 | 0 | 0000 | 10 | X | X | XXXX | 0 | 1 | 110 |

Assume register file 1 holds 3791FEF6 (Hex):

Source 0011 0111 1001 0001 1111 1110 1111 0110 R ← DA

Destination 1100 1000 0110 1110 0000 0001 0000 1010 RF(4) ← R' + Cn

FUNCTION

Evaluates $S' + Cn$.

DESCRIPTION

Data on the S bus is inverted and added to the carry. The result appears at the ALU and MQ shifters.

*The result of this instruction can be shifted in the same microcycle by specifying a shift instruction in the upper nibble (I7-I4) of the instruction field. The result may also be passed without shift. Possible instructions are listed in Table 15.

Available R Bus Source Operands

| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 : A3-A0 Mask |
|---------------|----------------|---------|-----------------------------|
| No | No | No | No |

Available S Bus Source Operands

| RF (B5-B0) | DB-Port | MQ Register |
|---------------|---------|----------------|
| Yes | Yes | Yes |

Available Destination Operands

| RF (C5-C0) | RF (B5-B0) | Y-Port | ALU Shifter | MQ Shifter |
|---------------|---------------|--------|----------------|---------------|
| Yes | No | Yes | Yes | Yes |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|----------------------|--|
| SSF | No | Affect shift instructions programmed in bits I7-I4 of instruction field. |
| $\overline{SIO0}$ | No | |
| $\overline{SIO1}$ | No | |
| $\overline{SIO2}$ | No | |
| $\overline{SIO3}$ | No | |
| Cn | Yes | Increments if programmed high. |

Status Signals[†]

| | |
|------|-----------------------------------|
| ZERO | = 1 if result = 0 |
| N | = 1 if MSB = 1 |
| OVR | = 1 if signed arithmetic overflow |
| C | = 1 if carry-out = 1 |

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Convert the data on the MQ register to one's complement and store the result in register 4.

| Instr Code | Oprd Addr | Oprd Addr | Oprd Sel EB1- EA EBO | Dest Addr | SEL | Destination Selects | | | | | | Cn | CF2- CF0 |
|---------------|--------------|--------------|----------------------------|--------------|-----|---------------------|-------------------|-------------|-------------|---------------|---------------|----|-------------|
| | | | | | | WE3- WE0 | SELRF1- SELRF0 | OE3- OE0 | OE1- OE0 | OEY3- OEY0 | OEY1- OEY0 | | |
| 1111 0101 | XX XXXX | XX XXXX | X 11 | 00 0100 | 0 | 0000 | 10 | X | X | XXXX | 0 | 0 | 110 |

Assume MQ register file 1 holds 3791FEF6 (Hex):

| | | |
|-------------|---|-----------------|
| Source | 0011 0111 1001 0001 1111 1110 1111 0110 | S ← MQ register |
| Destination | 1100 1000 0110 1110 0000 0001 0000 1001 | RF(4) ← S' + Cn |

FUNCTION

Increments R if the carry is set.

DESCRIPTION

Data on the R bus is added to the carry. The sum appears at the ALU and MQ shifters.

*The result of this instruction can be shifted in the same microcycle by specifying a shift instruction in the upper nibble (I7-I4) of the instruction field. The result may also be passed without shift. Possible instructions are listed in Table 15.

Available R Bus Source Operands

| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 A3-A0 Mask |
|---------------|----------------|---------|------------------------|
| Yes | No | Yes | No |

Available S Bus Source Operands (MSH)

| RF (B5-B0) | DB-Port | MQ Register |
|---------------|---------|----------------|
| No | No | No |

Available Destination Operands

| RF (C5-C0) | RF (B5-B0) | Y-Port | ALU Shifter | MQ Shifter |
|---------------|---------------|--------|----------------|---------------|
| Yes | No | Yes | Yes | Yes |

Control/Data Signals

| Signal | User Programmable | Use |
|--------------------------|-------------------|--|
| SSF | No | Affect shift instructions programmed in bits I7-I4 of instruction field. |
| $\overline{\text{SIO0}}$ | No | |
| $\overline{\text{SIO1}}$ | No | |
| $\overline{\text{SIO2}}$ | No | |
| $\overline{\text{SIO3}}$ | No | |
| Cn | Yes | Increments R if programmed high. |

Status Signals[†]

| | |
|------|-----------------------------------|
| ZERO | = 1 if result = 0 |
| N | = 1 if MSB = 1 |
| OVR | = 1 if signed arithmetic overflow |
| Cn | = 1 if carry-out = 1 |

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Increment the data on the DA bus and store the result in register 4.

| Instr Code I7-I0 | Oprd Addr A5-A0 | Oprd Addr B5-B0 | Oprd Sel EB1- EA EB0 | Dest Addr C5-C0 | SEL MQ | Destination Selects | | | | | | Cn | CF2- CF0 |
|------------------------|-----------------------|-----------------------|----------------------------|-----------------------|-----------|---------------------|-------------------|-------------|-------------|---------------|---------------|----|-------------|
| | | | | | | WE3- WE0 | SELRF1- SELRF0 | OE3- OE0 | OE1- OE0 | OEY3- OEY0 | OEY1- OEY0 | | |
| 1111 0110 | XX XXXX | XX XXXX | 1 XX | 00 0100 | 0 | 0000 | 10 | X | X | XXXX | 0 | 1 | 110 |

Assume register file 1 holds 3791FEF6 (Hex).

Source

0001 0111 1001 0001 1111 1110 1111 0110

R ← DA

Destination

0001 0111 1001 0001 1111 1110 1111 0111

RF(4) ← R + Cn

FUNCTION

Increments S if the carry is set.

DESCRIPTION

Data on the S bus is added to the carry. The sum appears at the ALU and MQ shifters.

*The result of this instruction can be shifted in the same microcycle by specifying a shift instruction in the upper nibble (I7-I4) of the instruction field. The result may also be passed without shift. Possible instructions are listed in Table 15.

Available R Bus Source Operands

| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 :: A3-A0 Mask |
|---------------|----------------|---------|------------------------------|
| No | No | No | No |

Available S Bus Source Operands

| RF (B5-B0) | DB-Port | MQ Register |
|---------------|---------|----------------|
| Yes | Yes | Yes |

Available Destination Operands

| RF (C5-C0) | RF (B5-B0) | Y-Port | ALU Shifter | MQ Shifter |
|---------------|---------------|--------|----------------|---------------|
| Yes | No | Yes | Yes | Yes |

Control/Data Signals

| Signal | User Programmable | Use |
|--------------------------|-------------------|--|
| SSF | No | Affect shift instructions programmed in bits I7-I4 of instruction field. |
| $\overline{\text{SIO0}}$ | No | |
| $\overline{\text{SIO1}}$ | No | |
| $\overline{\text{SIO2}}$ | No | |
| $\overline{\text{SIO3}}$ | No | |
| Cn | Yes | Increments S if programmed high. |

Status Signals†

| | |
|------|-----------------------------------|
| ZERO | = 1 if result = 0 |
| N | = 1 if MSB = 1 |
| OVR | = 1 if signed arithmetic overflow |
| C | = 1 if carry-out = 1 |

†C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Increment the data in the MQ register and store the result in register 4.

| Instr Code I7-I0 | Oprd Addr A5-A0 | Oprd Addr B5-B0 | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2- CF0 |
|------------------------|-----------------------|-----------------------|----------------------------|-----------------------|---------------------|-------------|-------------------|-------------|-------------|---------------|---------------|---------------|-----|-------------|
| | | | | | SELMQ | WE3- WE0 | SELRF1- SELRF0 | OE3- OE0 | OE1- OE0 | OEY3- OEY0 | OEY1- OEY0 | OEY3- OEY0 | | |
| 1111 0100 | XX XXXX | XX XXXX | X 11 | 00 0100 | 0 | 0000 | 10 | X | X | XXXX | 0 | 1 | 110 | |

Assume MQ register holds 54FF00FF (Hex):

Source 0101 0100 1111 1111 0000 0000 1111 1111 S ← MQ register

Destination 0101 0100 1111 1111 0000 0001 0000 0000 RF(4) ← S + Cn

FUNCTION

Load divide/BCD flip-flops from external data input.

DESCRIPTION

Uses an internal bypass path to load data from the S MUX directly into the divide/BCD flip-flops.

Available R Bus Source Operands

| RF (A5-A0) | A3-A0 Immed | DA-Port | C3-C0 : A3-A0 Mask |
|---------------|----------------|---------|-----------------------------|
| No | No | No | No |

Available S Bus Source Operands

| RF (B5-B0) | DB-Port | MQ Register |
|---------------|---------|----------------|
| Yes | Yes | Yes |

Available Destination Operands

| RF (C5-C0) | RF (B5-B0) | Y-Port | ALU Shifter | MQ Shifter |
|---------------|---------------|--------|----------------|---------------|
| No | No | No | No | No |

Control/Data Signals

| Signal | User Programmable | Use |
|--------------------------|----------------------|----------|
| SSF | No | Inactive |
| $\overline{\text{SIO0}}$ | No | Inactive |
| $\overline{\text{SIO1}}$ | No | Inactive |
| $\overline{\text{SIO2}}$ | No | Inactive |
| $\overline{\text{SIO3}}$ | No | Inactive |
| Cn | No | Inactive |

Status Signals

| | | |
|------|---|---|
| ZERO | = | 0 |
| N | = | 0 |
| OVR | = | 0 |
| C | = | 0 |

EXAMPLE (assumes a 32-bit configuration)

Load the divide/BCD flip-flops with data from the DB input bus.

| Instr Code | Oprd Addr | Oprd Addr | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2- CF0 |
|---------------|--------------|--------------|----------------------------|-----------------------|---------------------|-------------|-------------------|-------------|-------------|---------------|-------------|-------------|----|-------------|
| | | | | | SELMQ | WE3- WE0 | SELRF1- SELRF0 | OE3- OE0 | OE1- OE0 | OEY3- OEY0 | OE5- OE0 | OE7- OE0 | | |
| 0000 1111 | XX XXXX | XX XXXX | X 10 | XX XXXX | X | XXXX | XX | X | X | XXXX | X | X | X | 110 |

Assume DB input holds 2A08C618 (Hex):

Source

0010 1010 0000 1000 1100 0110 0001 1000

S ← DB bus

Destination

0010 1010 0000 1000 1100 0110 0001 1000

Divide/BCD flip-flops ← S

FUNCTION

Passes the result of the ALU instruction specified in the lower nibble of the instruction field to Y and the MQ register.

DESCRIPTION

The result of the arithmetic or logical operation specified in the lower nibble of the instruction field (I3-I0) is passed unshifted to Y and the MQ register.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

| ALU Shifter | MQ Shifter |
|-------------|------------|
| None | None |

Available Destination Operands

| RF (C5-C0) | RF (B5-B0) | Y-Port |
|---------------|---------------|--------|
| Yes | No | Yes |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|----------------------|-----------------|
| SSF | No | Holds MQ0 (LSB) |
| $\overline{SI00}$ | No | Inactive |
| $\overline{SI01}$ | No | Inactive |
| $\overline{SI02}$ | No | Inactive |
| $\overline{SI03}$ | No | Inactive |
| Cn | No | Inactive |

Status Signals[†]

| | |
|------|--|
| ZERO | = 1 if result = 0 |
| N | = 1 if MSB of result = 1 = 0 if MSB of result = 0 |
| OVR | = 1 if signed arithmetic overflow |
| C | = 1 if carry-out = 1 |

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Load the MQ register with data from register 1, and pass the data to the Y port.

(In this example, data is passed to the ALU by and INCR instruction without carry-in.)

| Instr Code I7-I0 | Oprd Addr A5-A0 | Oprd Addr B5-B0 | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF2- CF0 |
|------------------------|-----------------------|-----------------------|----------------------------|-----------------------|---------------------|----------------|-----------------|---------------|--------------|--------------|--------------|--------------|----|-------------|
| | | | | | WE3- SELMO | SELRF1- WEO | SELRF0- OEAE | OEY3- OEYO | OEY0- OES | OEY1- OEB | OEY2- OEA | OEY3- OEB | | |
| 1111 0110 | 00 0001 | XX XXXX | 0 XX | XX XXXX | 0 | XXXX | XX | X | X | XXXX | 0 | 0 | 0 | 110 |

Assume register file 1 holds 2A08C618 (Hex):

Source 0010 1010 0000 1000 1100 0110 0001 1000 $R \leftarrow RF(1)$

Destination 0010 1010 0000 1000 1100 0110 0001 1000 MQ register $\leftarrow R + Cn$

FUNCTION

Passes the result of the ALU instruction specified in the upper nibble of the instruction field to Y MUX. Performs a circular left shift on MQ.

DESCRIPTION

The result of the arithmetic or logical operation specified in the lower nibble of the instruction field (I3-I0) is passed unshifted to Y MUX.

The contents of the MQ register are rotated one bit to the left. The MSB is rotated out and passed to the LSB of the same word, which may be 1, 2, or 4 bytes long.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to the MQ register. If SSF is low, the MQ register will not be altered.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

| ALU Shifter | MQ Shifter |
|-------------|---------------|
| None | Circular Left |

Available Destination Operands (ALU Shifter)

| RF (C5-C0) | RF (B5-B0) | Y-Port |
|---------------|---------------|--------|
| Yes | No | Yes |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|-------------------|---|
| SSF | Yes | Passes shift result if high or floating; retains MQ without shift if low. |
| $\overline{SI}O0$ | No | Inactive |
| $\overline{SI}O1$ | No | Inactive |
| $\overline{SI}O2$ | No | Inactive |
| $\overline{SI}O3$ | No | Inactive |
| Cn | No | Affects arithmetic operation programmed in bits I3-I0 of instruction field. |

Status Signals†

| | |
|------|--|
| ZERO | = 1 if result = 0 |
| N | = 1 if MSB of result = 1 = 0 if MSB of result = 0 |
| OVR | = 1 if signed arithmetic overflow |
| C | = 1 if carry-out = 1 |

†C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Add data in register 1 to data on the DB bus with carry-in and store the unshifted result in register 1. Circular shift the contents of the MQ register one bit to the left.

| Instr | Oprd | Oprd | Oprd Sel | Dest | Destination Selects | | | | | | | | | |
|-----------|---------|---------|----------|---------|---------------------|---------|--------|-------|-----|------|-----|----|------|--|
| Code | Addr | Addr | EB1- | Addr | WE3- | SELRF1- | | OEY3- | | | | | CF2- | |
| I7-I0 | A5-A0 | B5-B0 | EA EBO | C5-C0 | SELMQ | WE0 | SELRFO | OEa | OEb | OEYO | OES | Cn | CF0 | |
| 1101 0001 | 00 0001 | XX XXXX | 0 10 | 00 0001 | 0 | 0000 | 10 | X | X | XXXX | 0 | 1 | 110 | |

Assume register file 1 holds 2508C618 (Hex), DB bus holds 11007530 (Hex), and MQ register holds 4DA99A0E (Hex).

 Source

0010 0101 0000 1000 1100 0110 0001 1000

R ← RF(1)

 Source

0001 0001 0000 0000 0111 0101 0011 0000

S ← DB bus

 Destination

0011 0110 0000 1001 0011 1011 0100 1001

RF(1) ← R + S + Cn

 Source

0100 1101 1010 1001 1001 1010 0000 1110

MQ shifter ← MQ register

 Destination

1001 1011 0101 0011 0011 0100 0001 1100

MQ register ← MQ shifter

FUNCTION

Passes the result of the ALU instruction specified in the upper nibble of the instruction field to Y MUX. Performs a left shift on MQ.

DESCRIPTION

The result of the arithmetic or logical operation specified in the lower nibble of the instruction field (I3-I0) is passed unshifted to Y MUX.

The contents of the MQ register are shifted one bit to the left. A zero is filled into the least significant bit of each word unless the \overline{SIO} input for that word is programmed low; this will force the least significant bit to one. The MSB is dropped from each word, which may be 1, 2, or 4 bytes long, depending on the configuration selected.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to the MQ register. If SSF is low, the MQ register will not be altered.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

| ALU Shifter | MQ Shifter |
|-------------|--------------|
| None | Logical Left |

Available Destination Operands (ALU Shifter)

| RF (C5-C0) | RF (B5-B0) | Y-Port |
|---------------|---------------|--------|
| Yes | No | Yes |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|-------------------|--|
| SSF | Yes | Passes shift result if high or floating; retains MQ without shift if low. |
| $\overline{SIO0}$ | Yes | Fills a zero in LSB of MQ shifter if high or floating; sets LSB to one if low. |
| $\overline{SIO1}$ | No | Inactive in 32-bit configuration; used in configurations to select end-fill in LSBs. |
| $\overline{SIO2}$ | No | |
| $\overline{SIO3}$ | No | |
| Cn | No | Affects arithmetic operation programmed in bits I3-I0 of instruction field. |

Status Signals[†]

| | |
|------|--|
| ZERO | = 1 if result = 0 |
| N | = 1 if MSB of result = 1 = 0 if MSB of result = 0 |
| OVR | = 1 if signed arithmetic overflow |
| C | = 1 if carry-out = 1 |

[†]C is ALU carry-out and is evaluated before shift operation. ZERO and N (negative) are evaluated after shift operation. OVR (overflow) is evaluated after ALU operation and after shift operation.

EXAMPLE (assumes a 32-bit configuration)

Add data in register 7 to data on the DB bus with carry-in and store the unshifted result in register 7. Shift the contents of the MQ register one bit to the left, filling a zero into the least significant bit.

| Instr Code I7-I0 | Oprd Addr A5-A0 | Oprd Addr B5-B0 | Oprd Sel EB1- EA EBO | Dest Addr C5-C0 | Destination Selects | | | | | | | | Cn | CF0 | SIO3- SIO0 | IESIO3- IESIO0 |
|------------------------|-----------------------|-----------------------|----------------------------|-----------------------|---------------------|------|--------|------|-----|-------|-----|---|-----|------|---------------|-------------------|
| | | | | | WE3- SELRF1- | | OEY3- | | | | | | | | | |
| | | | | | SELMQ | WEO | SELRFO | OEAE | OEB | OEOY0 | OES | | | | | |
| 1100 0001 | 00 0111 | XX XXXX | 0 10 | 00 0111 | 0 | 0000 | 10 | X | X | XXXX | 0 | 1 | 110 | 1111 | 0000 | |

Assume register file 7 holds 7308C618 (Hex), DB bus holds 54007530 (Hex), and MQ register holds 61A99A0E (Hex).

| | | |
|-------------|---|--------------------------|
| Source | 0111 0011 0000 1000 1100 0110 0001 1000 | R ← RF(7) |
| Source | 0101 0100 0000 0000 0111 0101 0011 0000 | S ← DB bus |
| Destination | 1100 0111 0000 1001 0011 1011 0100 1001 | RF(7) ← R + S + Cn |
| Source | 0110 0001 1010 1001 1001 1010 0000 1100 | MQ shifter ← MQ register |
| Destination | 1100 0011 0101 0011 0011 0100 0001 1000 | MQ register ← MQ shifter |

FUNCTION

Passes the result of the ALU instruction specified in the upper nibble of the instruction field to Y MUX. Performs an arithmetic right shift on MQ.

DESCRIPTION

The result of the arithmetic or logical operation specified in the lower nibble of the instruction field (I3-I0) is passed unshifted to Y MUX.

The contents of the MQ register are rotated one bit to the right. The sign bit of the most significant byte is retained. Bit 0 of the least significant byte is dropped.

The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to the MQ register. If SSF is low, the MQ register will not be altered.

*A list of ALU operations that can be used with this instruction is given in Table 15.

Shift Operations

| ALU Shifter | MQ Shifter |
|-------------|------------------|
| None | Arithmetic Right |

Available Destination Operands (ALU Shifter)

| RF (C5-C0) | RF (B5-B0) | Y-Port |
|---------------|---------------|--------|
| Yes | No | Yes |

Control/Data Signals

| Signal | User Programmable | Use |
|-------------------|-------------------|--|
| SSF | Yes | Passes shift result if high or floating; retains MQ without shift if low. |
| $\overline{SI00}$ | No | Outputs LSB of MQ shifter (inverted). |
| $\overline{SI01}$ | No | Inactive in 32-bit configurations; used in other configurations to output LSBs from MQ shifter (inverted). |
| $\overline{SI02}$ | No | |
| $\overline{SI03}$ | No | |
| Cn | No | Affects arithmetic operation programmed in bits I3-I0 of instruction field. |

FUNCTION

Passes the result of the ALU instruction specified in the upper nibble of the instruction field to Y MUX. Performs an arithmetic right shift on MO.

DESCRIPTION

The result of the arithmetic or logical operation specified in the lower nibble of the instruction field (3-10) is passed unshifted to Y MUX. The contents of the MO register are rotated one bit to the right. The sign bit of the most significant byte is retained. Bit 0 of the least significant byte is dropped. The shift may be made conditional on SSF. If SSF is high or floating, the shift result will be sent to the MO register. If SSF is low, the MO register will not be altered.

* A list of ALU operations that can be used with the instruction is given in Table 1.2.

Shift Operations

| ALU Shift | MO Shift |
|-----------|------------------|
| None | Arithmetic Right |

Available Destination Operands (ALU Shift)

| RF (CS-CO) | RF (BS-BO) | Y-Pol |
|------------|------------|-------|
| Yes | No | Yes |

Control/Status Signals

| Signal | User Programmable | Use |
|--------|-------------------|---|
| SSF | Yes | Passes shift result if high or floating; retains MO without shift if low. |
| SI00 | No | Outputs LSB of MO register (inverted). |
| SI01 | No | Inactive in 32-bit configurations; used in other configurations to output LSBs from MO register (inverted). |
| SI02 | No | Inverted. |
| SI03 | No | Affects arithmetic operation programmed in bits 13-10 of instruction field. |
| Cn | No | |

| | |
|---|-----------|
| General Information | 1 |
| SN74ACT8818A 16-Bit Microsequencer | 2 |
| SN74ACT8832A 32-Bit Registered ALU | 3 |
| SN74ACT8836A 32- × 32-Bit Parallel Multiplier | 4 |
| SN74ACT8841A Digital Crossbar Switch | 5 |
| SN74ACT8847 64-Bit Floating-Point/Integer Unit | 6 |
| SN74ACT8847 Application Information | 7 |
| SN74ACT8867 32-Bit Vector Processor Unit | 8 |
| Design Support | 9 |
| Mechanical Data | 10 |

| | |
|----|--|
| 1 | General Information |
| 2 | SN74ACT838A 18-Bit Microprocessor Unit |
| 3 | SN74ACT838A 32-Bit Microprocessor Unit |
| 4 | SN74ACT838A 32- x 32-Bit Parallel Multiplier |
| 5 | SN74ACT838A 16-Bit Counter/Divider |
| 6 | SN74ACT838A 64-Bit Floating-Point Divider Unit |
| 7 | SN74ACT838A Application Information |
| 8 | SN74ACT838A 32-Bit Vector Processor Unit |
| 9 | Power Dissipation |
| 10 | Physical Data |

SN74ACT8836A 32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

D3046, JANUARY 1988—REVISED JANUARY 1989

- Performs Full 32-Bit by 32-Bit Multiply/Accumulate in Flowthrough Mode in 48 ns (Min)
- Can be Pipelined for 30 ns (Min) Operation
- Performs 64-Bit by 64-Bit Multiplication in Five Cycles
- Supports Division Using Newton-Raphson Approximation
- Signed, Unsigned, or Mixed-Mode Multiply Operations
- Low-Power 1- μ m EPIC™ CMOS
- Multiplier, Multiplicand, and Product Can Be Complemented
- Accumulator Bypass Option
- TTL I/O Voltage Compatibility
- Three Independent 32-Bit Buses for Multiplicand, Multiplier, and Product
- Parity Generation/Checking
- Master/Slave Fault Detection
- Single 5-V Power Supply
- Optional Integer or Fractional Rounding

description

The 'ACT8836A is a 32-bit by 32-bit parallel multiplier/accumulator (MAC) suitable for low-power, high-speed operations in applications such as digital signal processing, array processing, and numeric data processing. High speed is achieved through the use of a Booth and Wallace Tree architecture.

Data is input to the chip through two registered 32-bit DA and DB input ports and output through a registered 32-bit Y output port. These registers have independent clock enable signals and can be made transparent for flowthrough operations.

The device can perform two's complement, unsigned, and mixed-data arithmetic. It can also operate as a 64-bit by 64-bit multiplier. Five clock cycles are required to perform a 64-bit by 64-bit multiplication and also multiplex the 128-bit result. Division is supported using Newton-Raphson approximation.

A multiply/accumulate mode is provided to add or subtract the accumulator from the product or the complement of the product. The accumulator is 67 bits wide to accommodate possible overflow. A warning flag (ETPERR) indicates whether overflow has occurred.

A rounding feature in the 'ACT8836A allows the result to be truncated or rounded to the nearest 32-bits. To ensure data integrity, byte parity checking is provided at the input ports, and a parity generator and master/slave error detection comparator are provided at the output port.

The 'ACT8836A is characterized for operation from 0°C to 70°C.

data flow

Two 32-bit input data ports, DA and DB, are provided for input of the multiplicand and multiplier to registers A and B and the multiplier/adder. Input data can be clocked to the A and B registers before being passed to the multiplier/adder if desired. Two multiplexers, R and S, in conjunction with a flowthrough decoder select the multiplier operands from DA and DB inputs, A and B registers, or the temporary register. Data is supplied to the temporary register from a shifter that operates on external DA/DB data or a previous multiplier/adder result. The 67-bit multiplier/adder result can be output through the Y port or passed through the shifter to the accumulator.

External DA and DB data is also available to the accumulator via the shifter. This 64-bit data can be extended with zeros or the sign bit. The 64 least significant bits from the shifter may also be latched in the 64-bit temporary register and input to the multiplier through the R and S multiplexers. A swap option allows the most significant and least significant 32-bit halves of temporary register data to be swapped before being made available to the R and S multiplexers. This allows either 32-bit half of the temporary register to be used as a multiplier.

EPIC is a trademark of Texas Instruments Incorporated

PRODUCTION DATA documents contain information current as of publication date. Products conform to specifications per the terms of Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.

TEXAS
INSTRUMENTS

POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

Copyright © 1989, Texas Instruments Incorporated

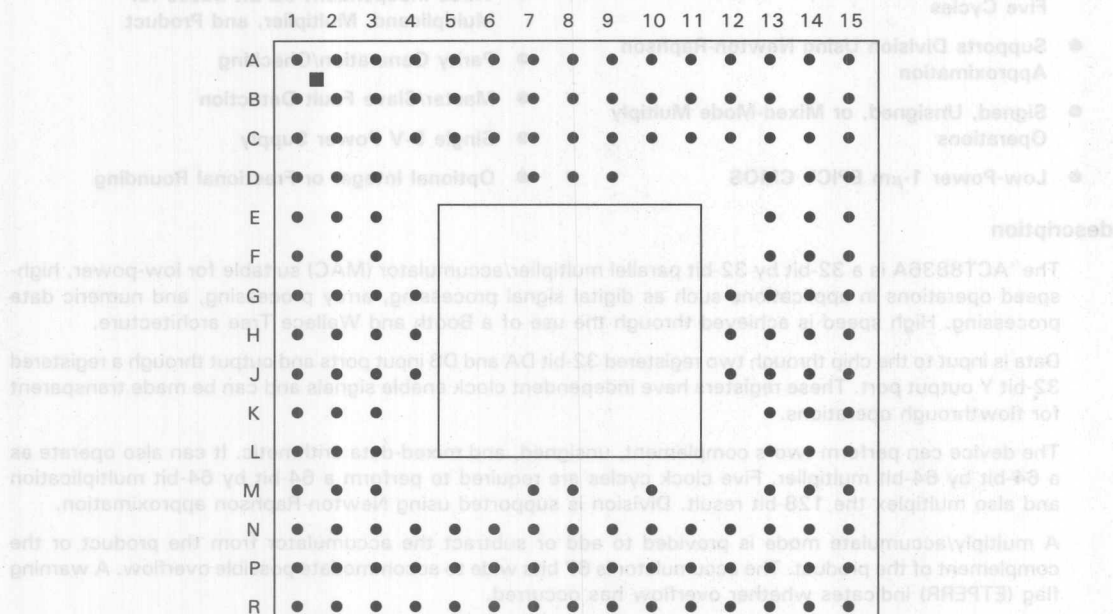
SN74ACT8836A **32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR**

pin descriptions

Pin assignments and terminal functions for the 'ACT8836A are given on the following pages. The pin at location M9 is omitted and a pin at M10 is added for indexing purposes.

15x15 GB PIN-GRID-ARRAY PACKAGE

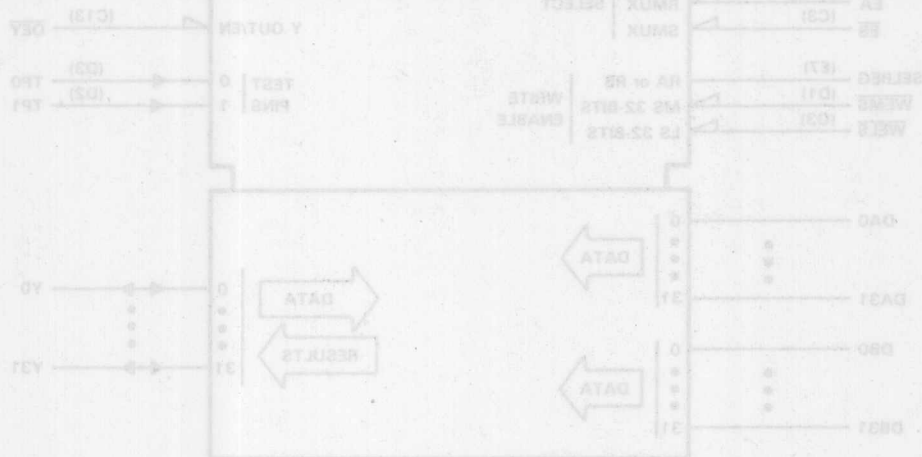
(TOP VIEW)



SN74ACT8836A
32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

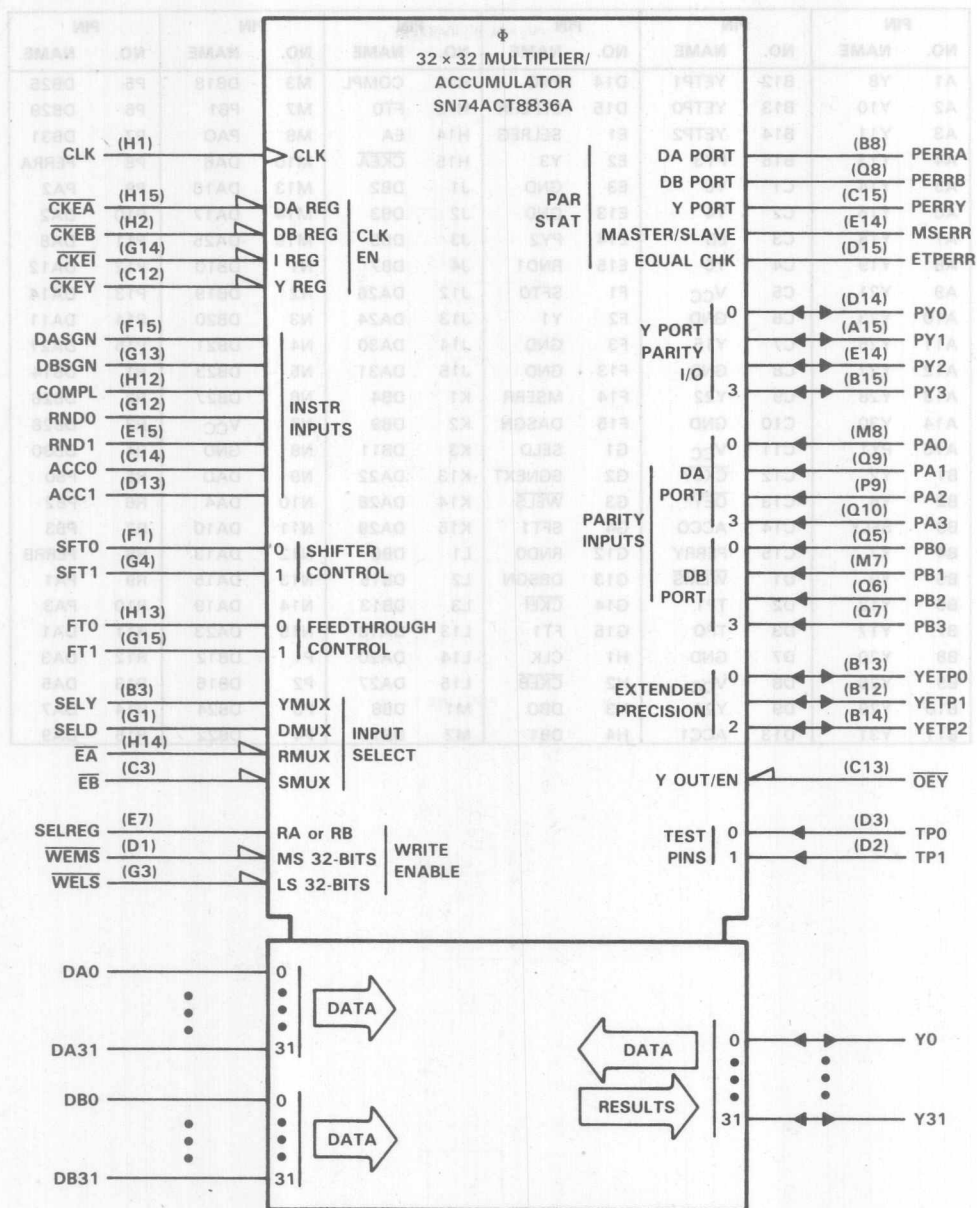
Pin Assignments

| PIN | | PIN | | PIN | | PIN | | PIN | |
|-----|------|-----|-------|-----|--------|-----|-------|-----|------|
| NO. | NAME | NO. | NAME | NO. | NAME | NO. | NAME | NO. | NAME |
| A1 | Y8 | B12 | YETP1 | D14 | PYO | H12 | COMPL | M3 | DB18 |
| A2 | Y10 | B13 | YETP0 | D15 | ETPERR | H13 | FT0 | M7 | PB1 |
| A3 | Y11 | B14 | YETP2 | E1 | SELREG | H14 | EA | M8 | PAO |
| A4 | Y13 | B15 | PY3 | E2 | Y3 | H15 | CKEA | M10 | DA6 |
| A5 | Y14 | C1 | Y0 | E3 | GND | J1 | DB2 | M13 | DA16 |
| A6 | Y16 | C2 | Y4 | E13 | GND | J2 | DB3 | M14 | DA17 |
| A7 | Y18 | C3 | EB | E14 | PY2 | J3 | DB5 | M15 | DA25 |
| A8 | Y19 | C4 | Y5 | E15 | RND1 | J4 | DB7 | N1 | DB10 |
| A9 | Y21 | C5 | VCC | F1 | SFT0 | J12 | DA26 | N2 | DB19 |
| A10 | Y23 | C6 | GND | F2 | Y1 | J13 | DA24 | N3 | DB20 |
| A11 | Y25 | C7 | Y15 | F3 | GND | J14 | DA30 | N4 | DB21 |
| A12 | Y27 | C8 | GND | F13 | GND | J15 | DA31 | N5 | DB23 |
| A13 | Y28 | C9 | Y22 | F14 | MSERR | K1 | DB4 | N6 | DB27 |
| A14 | Y30 | C10 | GND | F15 | DASGN | K2 | DB9 | N7 | VCC |
| A15 | PY1 | C11 | VCC | G1 | SELD | K3 | DB11 | N8 | GND |
| B1 | Y2 | C12 | CKEY | G2 | SGNEXT | K13 | DA22 | N9 | DAO |
| B2 | Y6 | C13 | OEY | G3 | WELS | K14 | DA28 | N10 | DA4 |
| B3 | SELY | C14 | ACCO | G4 | SFT1 | K15 | DA29 | N11 | DA10 |
| B4 | Y7 | C15 | PERRY | G12 | RND0 | L1 | DB6 | N12 | DA13 |
| B5 | Y9 | D1 | WEMS | G13 | DBSGN | L2 | DB15 | N13 | DA15 |
| B6 | Y12 | D2 | TP1 | G14 | CKEI | L3 | DB13 | N14 | DA19 |
| B7 | Y17 | D3 | TPO | G15 | FT1 | L13 | DA18 | N15 | DA23 |
| B8 | Y20 | D7 | GND | H1 | CLK | L14 | DA20 | P1 | DB12 |
| B9 | Y26 | D8 | VCC | H2 | CKEB | L15 | DA27 | P2 | DB16 |
| B10 | Y29 | D9 | Y24 | H3 | DBO | M1 | DB8 | P3 | DB24 |
| B11 | Y31 | D13 | ACC1 | H4 | DB1 | M2 | DB17 | P4 | DB22 |
| | | | | | | | | R15 | DA9 |



SN74ACT8836A **32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR**

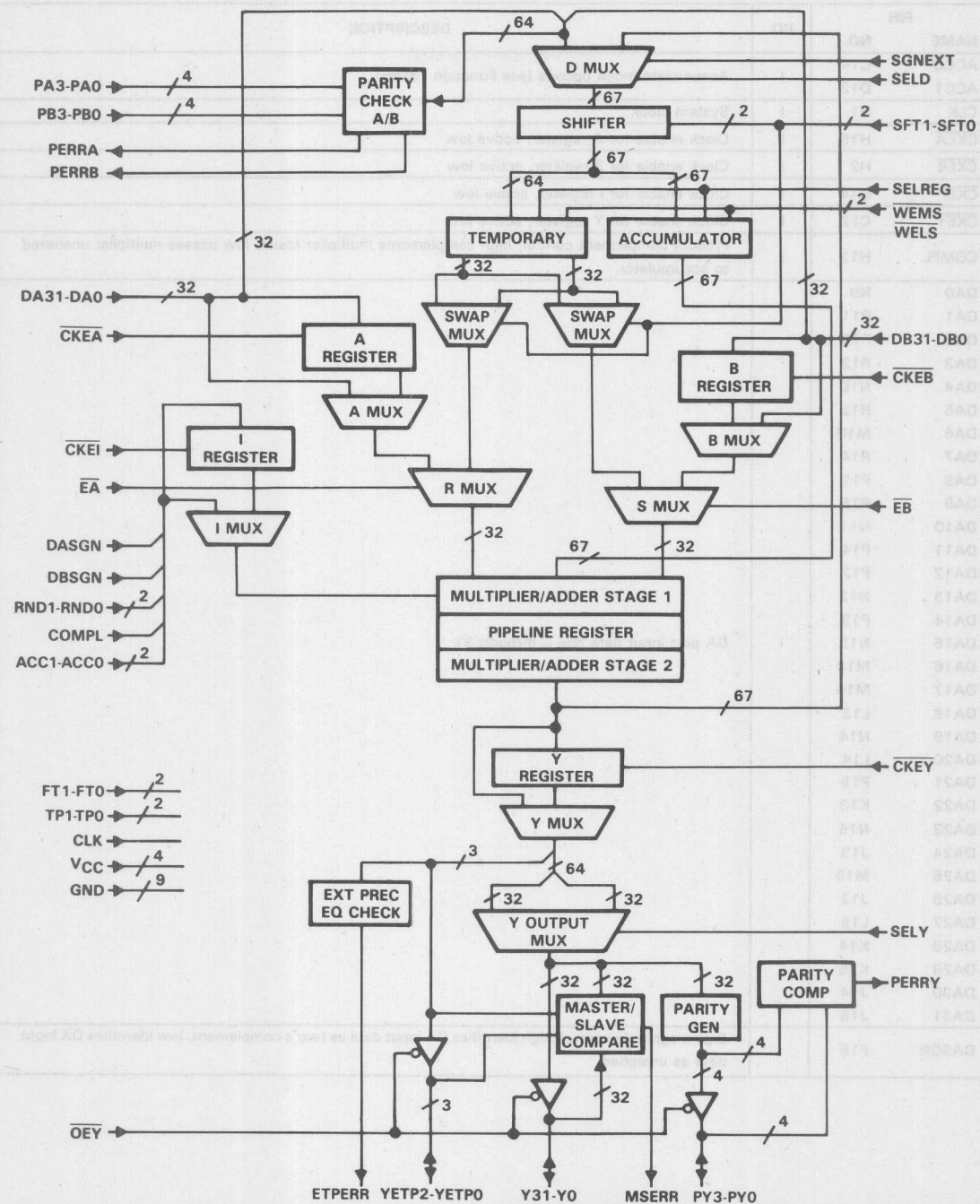
logic symbol†



†This symbol is in accordance with ANSI/IEEE-Std. 91-1984.

SN74ACT8836A
32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

functional block diagram (positive logic)



TEXAS
INSTRUMENTS

POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

SN74ACT8836A **32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR**

Terminal Functions

| PIN | | I/O | DESCRIPTION |
|-------|-----|-----|---|
| NAME | NO. | | |
| ACCO | C14 | I | Accumulate mode opcode (see Function Tables) |
| ACC1 | D13 | I | |
| CLK | H1 | I | System clock |
| CKEA | H15 | I | Clock enable for A register, active low |
| CKEB | H2 | I | Clock enable for B register, active low |
| CKEI | G14 | I | Clock enable for I register, active low |
| KEY | C12 | I | Clock enable for Y register, active low |
| COMPL | H12 | I | Product complement control; high complements multiplier result, low passes multiplier unaltered to accumulator. |
| DA0 | N9 | I | DA port input data bits 0 through 31 |
| DA1 | R11 | | |
| DA2 | P10 | | |
| DA3 | R12 | | |
| DA4 | N10 | | |
| DA5 | R13 | | |
| DA6 | M10 | | |
| DA7 | R14 | | |
| DA8 | P11 | | |
| DA9 | R15 | | |
| DA10 | N11 | | |
| DA11 | P14 | | |
| DA12 | P12 | | |
| DA13 | N12 | | |
| DA14 | P13 | | |
| DA15 | N13 | | |
| DA16 | M13 | | |
| DA17 | M14 | | |
| DA18 | L13 | | |
| DA19 | N14 | | |
| DA20 | L14 | | |
| DA21 | P15 | | |
| DA22 | K13 | | |
| DA23 | N15 | | |
| DA24 | J13 | | |
| DA25 | M15 | | |
| DA26 | J12 | | |
| DA27 | L15 | | |
| DA28 | K14 | | |
| DA29 | K15 | | |
| DA30 | J14 | | |
| DA31 | J15 | | |
| DASGN | F15 | I | Sign magnitude control; high identifies DA input data as two's complement, low identifies DA input data as unsigned |

Terminal Functions (Continued)

| NAME | PIN NO. | I/O | DESCRIPTION |
|--------|---------|-----|---|
| DB0 | H3 | | |
| DB1 | H4 | | |
| DB2 | J1 | | |
| DB3 | J2 | | |
| DB4 | K1 | | |
| DB5 | J3 | | |
| DB6 | L1 | | |
| DB7 | J4 | | |
| DB8 | M1 | | |
| DB9 | K2 | | |
| DB10 | N1 | | |
| DB11 | K3 | | |
| DB12 | P1 | | |
| DB13 | L3 | | |
| DB14 | R1 | | |
| DB15 | L2 | I | DB port input data bits 0 through 31 |
| DB16 | P2 | | |
| DB17 | M2 | | |
| DB18 | M3 | | |
| DB19 | N2 | | |
| DB20 | N3 | | |
| DB21 | N4 | | |
| DB22 | P4 | | |
| DB23 | N5 | | |
| DB24 | P3 | | |
| DB26 | R2 | | |
| DB27 | N6 | | |
| DB28 | R3 | | |
| DB29 | P6 | | |
| DB30 | R4 | | |
| DB31 | P7 | | |
| DBSGN | G13 | I | Sign magnitude control; high identifies DB input data as two's complement, low identifies DB input data as unsigned. |
| EA | H14 | I | Core multiplier operand select. A high on this signal selects DA register for input on the R bus; a low selects the swap MUX. |
| EB | C3 | I | Core multiplier operand select. A high on this signal selects DB register for input on the S bus; a low selects the swap MUX. |
| ETPERR | D15 | O | Equality check result. A low on this signal indicates that bits 67 through 64 of the core multiplier results are equal to bit 63. |
| FT0 | H13 | I | Feedthrough control signals for A, B, I, Pipeline and Y registers (see Function Tables). |
| FT1 | G15 | | |

SN74ACT8836A **32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR**

Terminal Functions (Continued)

| NAME | PIN NO. | I/O | DESCRIPTION |
|------------------|---------|-----|--|
| GND | C6 | | Ground pins. All ground pins should be used and connected. |
| GND | C8 | | |
| GND | C10 | | |
| GND | D7 | | |
| GND | E3 | | |
| GND | E13 | | |
| GND | F3 | | |
| GND | F13 | | |
| GND | N8 | | |
| MSERR | F14 | O | Master/slave error flag. This signal goes high when the contents of the Y output multiplexer and the value at the external port are not equal. |
| \overline{OEY} | C13 | I | Y, YETP2-YETP0, and PY3-PY0 output enable, active low. |
| PA0 | M8 | I | Parity input data bus for DA input data |
| PA1 | R9 | | |
| PA2 | P9 | | |
| PA3 | R10 | | |
| PB0 | R5 | I | Parity input data bus for DB input data |
| PB1 | M7 | | |
| PB2 | R6 | | |
| PB3 | R7 | | |
| PY0 | D14 | I/O | Y output parity data bus. Outputs data from parity generator ($\overline{OEY} = L$) or inputs external parity data ($\overline{OEY} = H$). |
| PY1 | A15 | | |
| PY2 | E14 | | |
| PY3 | B15 | | |
| PERRA | P8 | O | DA port parity status pin. Goes high if even-parity test on any byte fails. |
| PERRB | R8 | O | DB port parity status pin. Goes high if even-parity test on any byte fails. |
| PERRY | C15 | O | Y port parity status pin. Goes high if even-parity test on any byte fails. |
| RND0 | G12 | I | Multiplier/accumulator rounding control; high rounds integer result; low leaves result unaltered. |
| RND1 | E15 | I | Multiplier/accumulator rounding control; high rounds fractional result; low leaves result unaltered. |
| SELD | G1 | I | D multiplexer select. High selects DA and DB ports; low selects multiplier core output. |
| SELREG | E1 | I | Write enable for temporary register and accumulator. High enables the temporary register; low enables the accumulator. |
| SELY | B3 | I | Y multiplexer select. High selects most significant 32 bits of Y register output; low selects least significant 32 bits. |
| SGNEXT | G2 | I | Sign extend control for multiplexer. A low fills shift matrix bits 66-64 with zeros; a high fills DA31 in bits 66-64. |
| SFT0 | F1 | I | Shift multiplexer control (see Function Tables). |
| SFT1 | G4 | I | |
| TPO | D3 | I | Test pins (see Function Tables) |
| TP1 | D2 | | |
| VCC | C5 | | Supply voltage (5 V) |
| VCC | C11 | | |
| VCC | D8 | | |
| VCC | N7 | | |
| WEMS | D1 | I | Write enable for most significant 32 bits of temporary register and accumulator active low. |
| WELS | G3 | I | Write enable for least significant 32 bits of temporary register and accumulator active low. |



POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

Terminal Functions (Continued)

| NAME | PIN NO. | I/O | DESCRIPTION |
|-------|---------|-----|---|
| Y0 | C1 | I/O | <p>Y port data bus. Outputs data from Y register ($\overline{OEY} = L$); inputs data to master/slave comparator ($\overline{OEY} = H$).</p> |
| Y1 | F2 | | |
| Y2 | B1 | | |
| Y3 | E2 | | |
| Y4 | C2 | | |
| Y5 | C4 | | |
| Y6 | B2 | | |
| Y7 | B4 | | |
| Y8 | A1 | | |
| Y9 | B5 | | |
| Y10 | A2 | | |
| Y11 | A3 | | |
| Y12 | B6 | | |
| Y13 | A4 | | |
| Y14 | A5 | | |
| Y15 | C7 | | |
| Y16 | A6 | | |
| Y17 | B7 | | |
| Y18 | A7 | | |
| Y19 | A8 | | |
| Y20 | B8 | | |
| Y21 | A9 | | |
| Y22 | C9 | | |
| Y23 | A10 | | |
| Y24 | D9 | | |
| Y25 | A11 | | |
| Y26 | B9 | | |
| Y27 | A12 | | |
| Y28 | A13 | | |
| Y29 | B10 | | |
| Y30 | A14 | | |
| Y31 | B11 | | |
| YETP0 | B13 | I/O | Data bus for extended precision product. Outputs three most significant bits of the 67-bit multiplier core result; inputs external data to master/slave comparator. |
| YETP1 | B12 | | |
| YETP2 | B14 | | |

SN74ACT8836A
32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

Function Tables

MULTIPLIER/ADDER CONTROL INPUTS

| ACC1 | ACC0 | EA | EB | Operation |
|------|------|----|----|--------------------------------------|
| 0 | 0 | X | X | $\pm(R \times S) + 0$ |
| 0 | 1 | X | X | $\pm(R \times S) + \text{ACC}$ |
| 1 | 0 | X | X | $\pm(R \times S) - \text{ACC}$ |
| 1 | 1 | 0 | 0 | $\pm 1 \times 1 + 0$ |
| 1 | 1 | 0 | 1 | $\pm 1 \times \text{DB} + 0$ |
| 1 | 1 | 1 | 0 | $\pm \text{DA} \times 1 + 0$ |
| 1 | 1 | 1 | 1 | $\pm \text{DA} \times \text{DB} + 0$ |

ACC is the data stored in the accumulator

SHIFTER CONTROL INPUTS

| SFT1 | SFT0 | Shifter Operation |
|------|------|---|
| L | L | Pass data without shift |
| L | H | Shift one bit left; fill with zero |
| H | L | Swap upper and lower halves of temporary register |
| H | H | Shift 32 bits right; fill with sign bit |

FLOWTHROUGH CONTROL INPUTS

| Control Inputs | | Registers Bypassed | | | | |
|----------------|-----|--------------------|-----|-----|-----|-----|
| FT1 | FT0 | Pipeline | Y | I | A | B |
| L | L | Yes | Yes | Yes | Yes | Yes |
| L | H | Yes | No | No | No | No |
| H | L | Yes | Yes | No | No | No |
| H | H | No | No | No | No | No |

TEST PIN CONTROL INPUTS

| TP1 | TP0 | Operation |
|-----|-----|--|
| L | L | All outputs and I/Os forced low |
| L | H | All outputs and I/Os forced high |
| H | L | All outputs placed in a high impedance state |
| H | H | Normal operation (default state) |

architectural elements

Included in the functional block diagram of the 'ACT8836A are the following blocks.

1. Two 32-bit registered input data ports DA and DB
2. A parity checker at the DA and DB inputs
3. An instruction decoder (I register)
4. A flowthrough decoder that permits selected registers to be bypassed to support up to three levels of pipelining
5. R and S multiplexers to select operands for the multiplier/ adder from DA and DB inputs, registers A and B, or temporary register
6. A D multiplexer that selects the operand for the shifter from the 67-bit sign-extended DA and DB inputs or the multiplier/adder output
7. A shifter block that operates on DA/DB input data or on multiplier/adder outputs for scaling or Newton-Raphson division
8. A Y output multiplexer that selects the most significant half or the least significant half of the multiplier/ adder result for output at the registered Y port
9. An extended precision error check that tests for overflow
10. A master/slave comparator and parity generator/comparator at the Y output port for master/slave and parity checking
11. Registers at the external data and instruction input ports and the shifter and multiplier/adder output port to support pipelining

input data parity checker

An even-parity check is performed on each byte of input data at the DA, DB and Y ports. If the parity test fails for any byte, a high appears at the parity error output pin (PERRA for DA data, PERRB for DB data, PERRY for Y data).

A and B registers

Register A can be loaded with data from the DA bus, which normally holds a 32-bit multiplicand. Register B is loaded from the DB bus which holds a 32-bit multiplier. Separate clock enables, $\overline{\text{CKEA}}$ and $\overline{\text{CKEB}}$, allow the registers to be loaded separately. This is useful when performing double precision multiplication or using the temporary register as an input to the multiplier/adder. The registers can be made transparent using the FT inputs (see Function Tables).

instruction register

Instruction inputs to the device are shown in Table 1. These signals control signed, unsigned, and mixed multiplication modes, fractional and integer rounding, accumulator operations, and complementing of products. They can be latched into instruction register I when clock enable $\overline{\text{CKEI}}$ is low.

TABLE 1. INSTRUCTION INPUTS

| Signal | High | Low |
|--------------|--|---|
| DASGN | Identifies DA input data as two's complement | Identifies DA input data as unsigned |
| DBSGN | Identifies DB input data as two's complement | Identifies DB input data as unsigned |
| RND0 | Rounds integer result | Leaves integer result unaltered |
| RND1 | Rounds fractional result | Leaves fractional result unaltered |
| COMPL | Complements the product from the multiplier before passing it to the accumulator | Passes the product from the multiplier to the accumulator unaltered |
| ACC0 ACC1 | See Function Tables | See Function Tables |

Sign control inputs DASGN and DBSGN identify DA and DB input data as signed (high) or unsigned (low).

SN74ACT8836A **32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR**

Rounding inputs RND0 and RND1 control rounding operations in the multiplier/adder. A low on these inputs passes the results unaltered. If a high appears on RND1, the result will be rounded by adding a 1 to bit 30. RND1 should be set high if the multiplier/adder result is to be shifted in order to maintain precision of the least significant bit following the shift operation. If a high appears on RND0, the result will be rounded by adding a one to bit 31. This code should be used when the adder result will not be shifted.

A complement control, COMPL, is used to complement the product from the multiplier before passing it to the accumulator. The complement will occur if COMPL is high; the product will be passed unaltered if COMPL is low.

ACC1-ACC0 control the operation of the multiplier/adder. Possible operations are shown in the function tables.

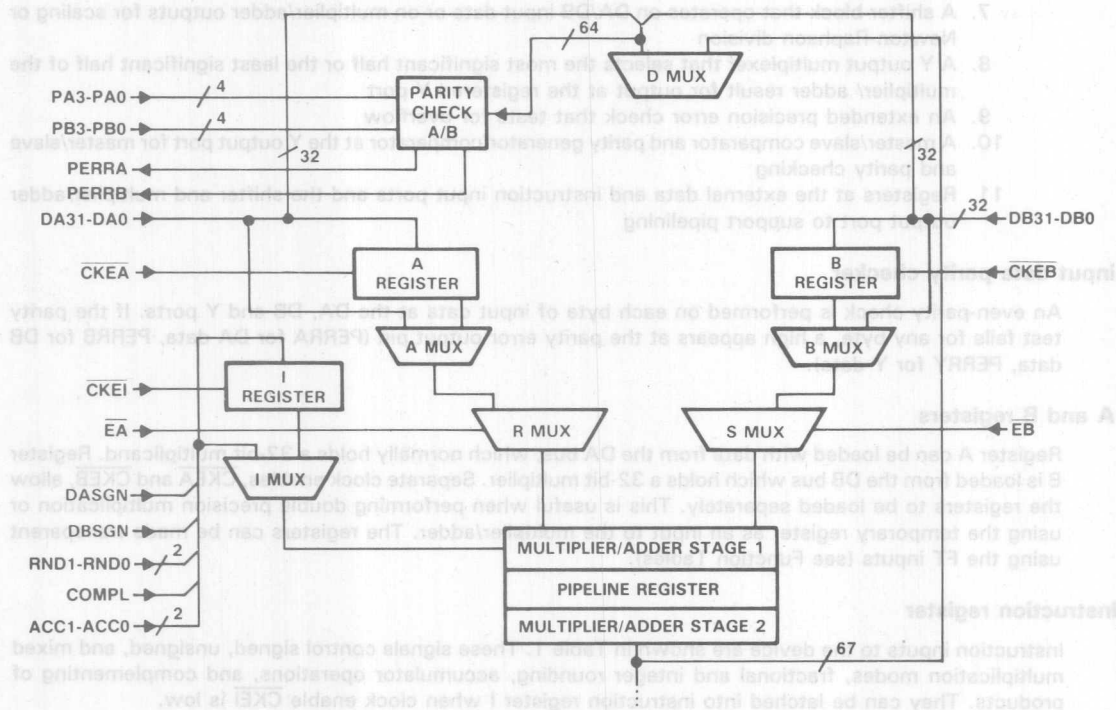


FIGURE 1. INPUT REGISTERS AND PARITY CHECK

| Signal | High | Low |
|--------|---|--|
| ACC1 | See Function Table | See Function Table |
| ACC0 | See Function Table | See Function Table |
| COMPL | Complements the product from the multiplier before passing it to the accumulator. | Passes the product from the multiplier to the accumulator unaltered. |
| RND1 | Rounds fractional result. | Leaves fractional result unaltered. |
| RND0 | Rounds integer result. | Leaves integer result unaltered. |
| DASGN | Identifies DA input data as two's complement. | Identifies DA input data as unsigned. |
| DBSGN | Identifies DB input data as two's complement. | Identifies DB input data as unsigned. |

R, S, and swap multiplexers

The R and S multiplexers select the multiplier/adder operands from external data or from the temporary register.

When \overline{EA} is low, the R multiplexer selects data from the swap multiplexer. When \overline{EA} is high, the R multiplexer selects data from DA or the A register, depending on the state of the flowthrough control inputs (see Function Tables). When \overline{EB} is low, the S multiplexer selects data from the swap multiplexer. When \overline{EB} is high, the S multiplexer switches data from DB or the B register, depending on the state of the flowthrough control inputs.

\overline{EA} and \overline{EB} are also used in conjunction with the multiplier/adder control inputs to force a numeric one on the R or S inputs (see Function Tables).

The swap multiplexers are controlled by the shifter control inputs. When SFT1 is high and SFT0 is low, the most significant half of the temporary register is available to the S multiplexer, and the least significant half is available to the R multiplexer. When SFT1-SFT0 are set to other values, the most significant half of the temporary register is available to the R multiplexer, and the least significant half is available to the S multiplexer.

multiplier/adder

The multiplier performs 32-bit multiplication and generates a 67-bit product. The product can be latched in the pipeline to increase cycle speed. The product is complemented when COMPL is set high as shown in Table 1. The adder computes the sum or the difference of the accumulator and the product and gives a 67-bit sum. Bits 66-64 are used for overflow and sign extension.

D multiplexer

The D multiplexer selects input data for the shifter. Two sources are available to the multiplexer: a 64-bit word formed by concatenating DA and DB bus data, and the 67-bit sum from the multiplier/adder. If SELD is high, external DA/DB data is selected; if SELD is low, the sum is selected.

If the 64-bit word is selected for input to the shifter, three bits are added to the word based on the state of the sign extend signal (SGNEXT). If SGNEXT is low, bits 66-64 are zero-filled; if SGNEXT is high, bits 66-64 are filled with the value on DA31.

temporary register and accumulator

Output from the shifter will be stored in the temporary register if SELREG is high and in the accumulator register if SELREG is low. The 64-bit temporary register can be used to store temporary data, constants, and scaled binary fractions (see Figure 2).

Separate clock controls, \overline{WELS} and \overline{WEMS} , allow the most significant and least significant halves of the shifter output to be loaded separately. The 32 least significant bits of the selected register are loaded when \overline{WELS} is low; the most significant bits when \overline{WEMS} is low. When \overline{WELS} and \overline{WEMS} are both low, the entire word from the shifter is loaded into the selected register.

shifter

The shifter can be used to multiply by two for Newton-Raphson operations or perform a 32-bit shift for double precision multiplication. The shifter is controlled by two SFT inputs, as shown in the function tables.

Y register

Final or intermediate multiplier/adder results will be clocked into Y register when \overline{CKEY} is low. Results can be passed directly to the Y output multiplexer using flowthrough decoder signals to bypass the register (see Function Tables).

SN74ACT8836A
32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

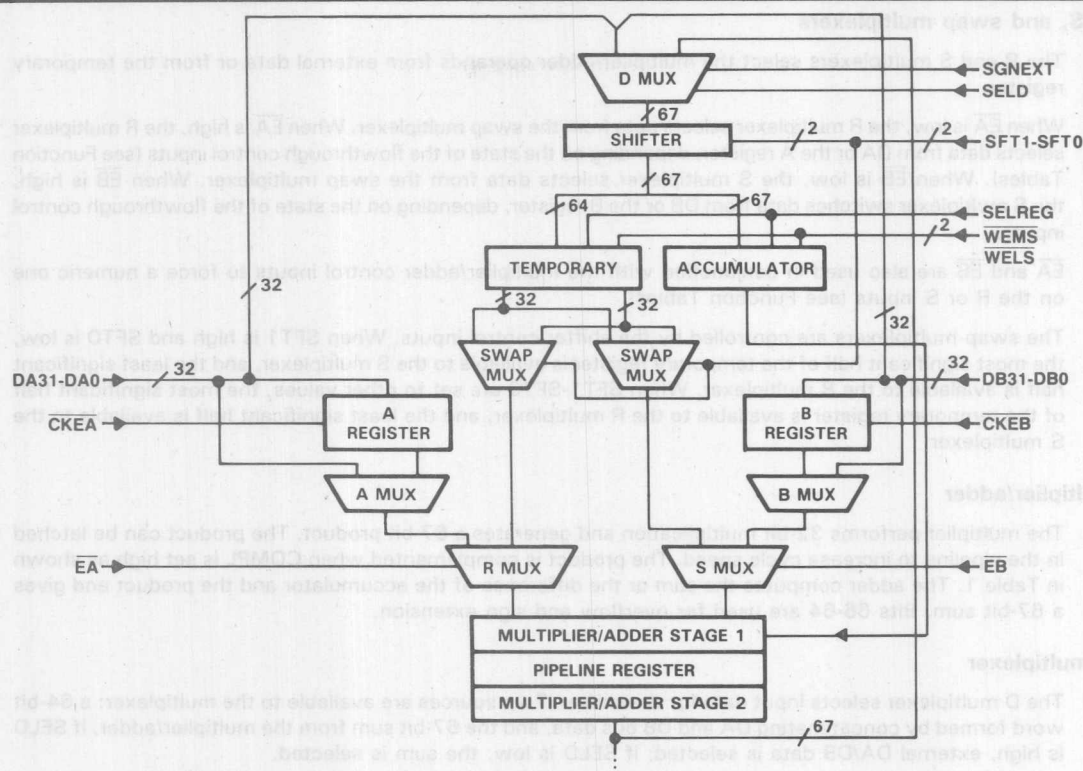


FIGURE 2. TEMPORARY REGISTER AND ACCUMULATION

Y multiplexer and Y output multiplexer

The Y multiplexer allows the 64-bit result or the contents of the Y register to be switched to the Y bus, depending upon the state of the flowthrough control outputs. The upper 32 bits are selected for output when the Y output multiplexer control SELY is high; the lower 32 bits are selected for output when SELY is low. Note that the Y output multiplexer can be switched at twice the clock rate so that the 64-bit result can be output in one clock cycle.

flowthrough decoder

To enable the device to operate in pipelined or flowthrough modes, on-chip registers can be bypassed using flowthrough control signals FT1 and FT0. Up to three levels of pipeline can be supported, as shown in the function tables.

extended precision check

Three extended product outputs, YETP2-YETP0, are provided to recover three bits of precision during overflow. An extended precision check error signal (ETPERR) goes high whenever overflow occurs. If sign controls DASGN and DBSGN are both low, indicating an unsigned operation, the extended precision bits 66-64 are compared for equality. Under all other sign control conditions, bits 66-63 are compared for equality.



SN74ACT8836A 32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

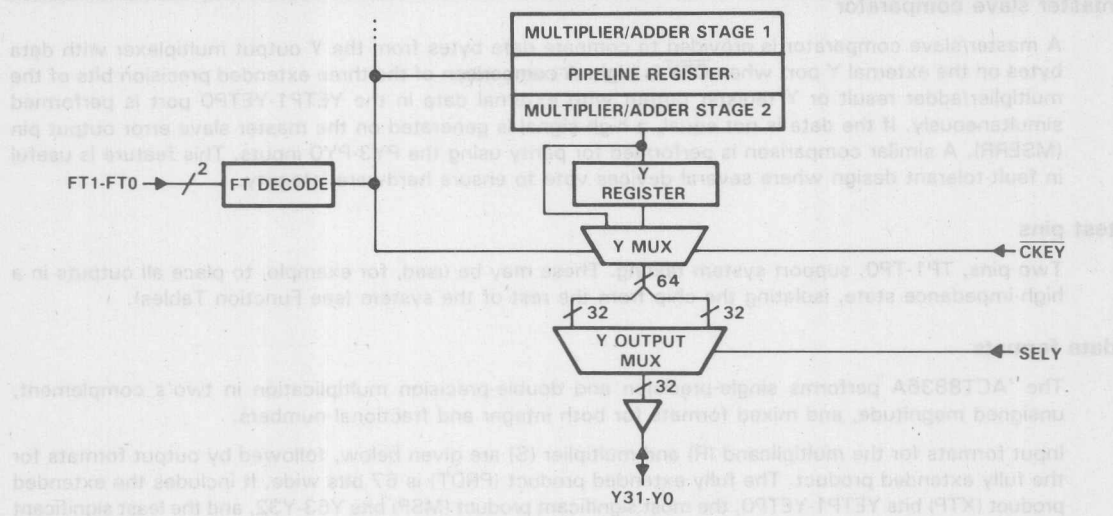


FIGURE 3. OUTPUT

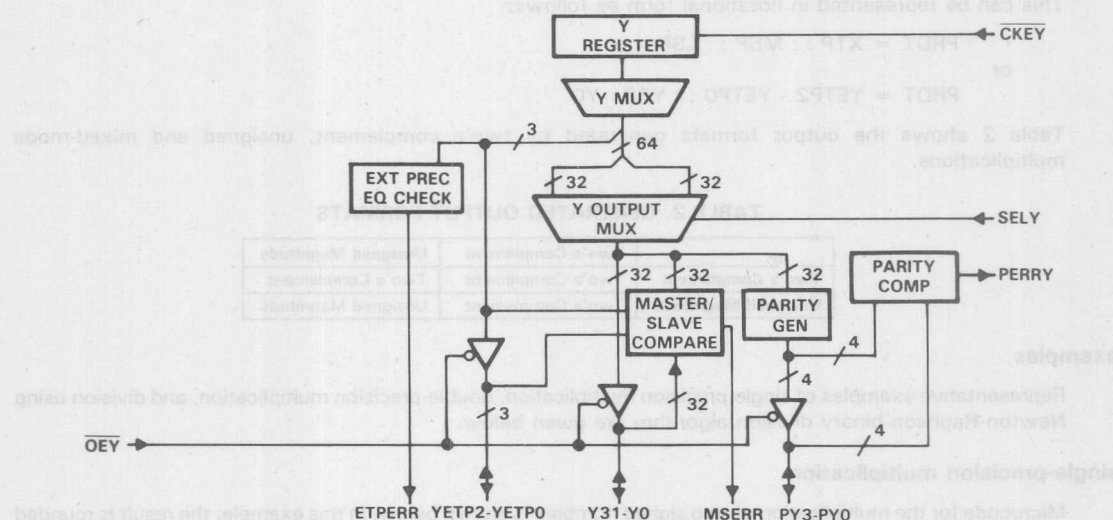


FIGURE 4. OUTPUT ERROR CONTROL

extended precision check

Three extended product outputs, YETP2-YETP0, are provided to recover three bits of precision during overflow. An extended precision check error signal (ETPERR) goes high whenever overflow occurs. If sign controls DASGN and DBSGN are both low, indicating an unsigned operation, the extended precision bits 66-64 are compared for equality. Under all other sign control conditions, bits 66-63 are compared for equality.

SN74ACT8836A
32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

master slave comparator

A master/slave comparator is provided to compare data bytes from the Y output multiplexer with data bytes on the external Y port when \overline{OEY} is high. A comparison of the three extended precision bits of the multiplier/adder result or Y register output with external data in the YETP1-YETP0 port is performed simultaneously. If the data is not equal, a high signal is generated on the master slave error output pin (MSERR). A similar comparison is performed for parity using the PY3-PY0 inputs. This feature is useful in fault-tolerant design where several devices vote to ensure hardware integrity.

test pins

Two pins, TP1-TP0, support system testing. These may be used, for example, to place all outputs in a high-impedance state, isolating the chip from the rest of the system (see Function Tables).

data formats

The 'ACT8836A performs single-precision and double-precision multiplication in two's complement, unsigned magnitude, and mixed formats for both integer and fractional numbers.

Input formats for the multiplicand (R) and multiplier (S) are given below, followed by output formats for the fully extended product. The fully extended product (PRDT) is 67 bits wide. It includes the extended product (XTP) bits YETP1-YETP0, the most significant product (MSP) bits Y63-Y32, and the least significant product (LSP) bits Y31-Y0.

This can be represented in notational form as follows:

$$\text{PRDT} = \text{XTP} :: \text{MSP} :: \text{LSP}$$

or
$$\text{PRDT} = \text{YETP2} - \text{YETP0} :: \text{Y63} - \text{Y0}$$

Table 2 shows the output formats generated by two's complement, unsigned and mixed-mode multiplications.

TABLE 2. GENERATED OUTPUT FORMATS

| | Two's Complement | Unsigned Magnitude |
|--------------------|------------------|--------------------|
| Two's Complement | Two's Complement | Two's Complement |
| Unsigned Magnitude | Two's Complement | Unsigned Magnitude |

examples

Representative examples of single-precision multiplication, double-precision multiplication, and division using Newton-Raphson binary division algorithm are given below.

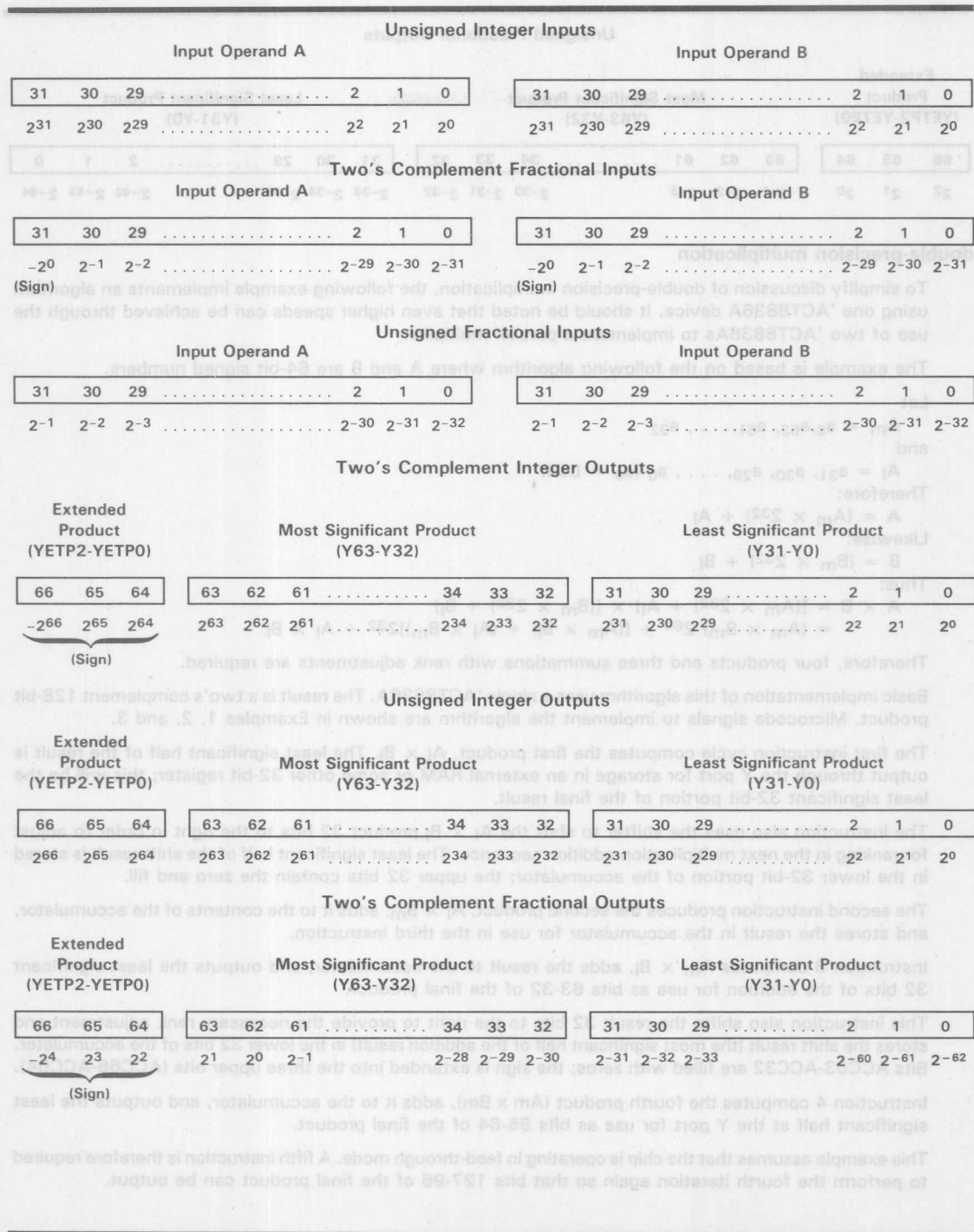
single-precision multiplication

Microcode for the multiplication of two signed numbers is shown below. In this example, the result is rounded and the 32 most significant bits are output on the Y bus. A second instruction ($\text{SELY} = 0$) would be required to output the least significant half if rounding were not used.

Unsigned and mixed mode single-precision multiplication are executed using the same code. (The sign controls must be modified accordingly.) Following are the input and output formats for signed, unsigned, and mixed mode operations.

| Two's Complement Integer Inputs | | | | | | | | | | | | | |
|---------------------------------|-----------------|-----------------|-------|----------------|----------------|----------------|------------------|-----------------|-----------------|-------|----------------|----------------|----------------|
| Input Operand A | | | | | | | Input Operand B | | | | | | |
| 31 | 30 | 29 | | 2 | 1 | 0 | 31 | 30 | 29 | | 2 | 1 | 0 |
| -2 ³¹ | 2 ³⁰ | 2 ²⁹ | | 2 ² | 2 ¹ | 2 ⁰ | -2 ³¹ | 2 ³⁰ | 2 ²⁹ | | 2 ² | 2 ¹ | 2 ⁰ |
| (Sign) | | | | | | | (Sign) | | | | | | |

SN74ACT8836A 32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR



SN74ACT8836A **32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR**

Unsigned Fractional Outputs

| Extended Product (YETP2-YETP0) | | | Most Significant Product (Y63-Y32) | | | | | | Least Significant Product (Y31-Y0) | | | | | |
|-----------------------------------|----------------|----------------|---------------------------------------|-----------------|-----------------|------------------|------------------|------------------|---------------------------------------|------------------|------------------|------------------|------------------|------------------|
| 66 | 65 | 64 | 63 | 62 | 61 | 34 | 33 | 32 | 31 | 30 | 29 | 2 | 1 | 0 |
| 2 ² | 2 ¹ | 2 ⁰ | 2 ⁻¹ | 2 ⁻² | 2 ⁻³ | 2 ⁻³⁰ | 2 ⁻³¹ | 2 ⁻³² | 2 ⁻³³ | 2 ⁻³⁴ | 2 ⁻³⁵ | 2 ⁻⁶² | 2 ⁻⁶³ | 2 ⁻⁶⁴ |

double-precision multiplication

To simplify discussion of double-precision multiplication, the following example implements an algorithm using one 'ACT8836A device. It should be noted that even higher speeds can be achieved through the use of two 'ACT8836As to implement a parallel multiplier.

The example is based on the following algorithm where A and B are 64-bit signed numbers.

Let

$$A_m = a_{63}, a_{62}, a_{61}, \dots, a_{32}$$

and

$$A_l = a_{31}, a_{30}, a_{29}, \dots, a_0 \text{ (} a_0 = \text{LSB)}$$

Therefore:

$$A = (A_m \times 2^{32}) + A_l$$

Likewise:

$$B = (B_m \times 2^{32}) + B_l$$

Thus:

$$\begin{aligned} A \times B &= [(A_m \times 2^{32}) + A_l] \times [(B_m \times 2^{32}) + B_l] \\ &= (A_m \times B_m) 2^{64} + [(A_m \times B_l) + (A_l \times B_m)] 2^{32} + A_l \times B_l \end{aligned}$$

Therefore, four products and three summations with rank adjustments are required.

Basic implementation of this algorithm uses a single 'ACT8836A. The result is a two's complement 128-bit product. Microcode signals to implement the algorithm are shown in Examples 1, 2, and 3.

The first instruction cycle computes the first product, $A_l \times B_l$. The least significant half of the result is output through the Y port for storage in an external RAM or some other 32-bit register; this will be the least significant 32-bit portion of the final result.

The instruction also uses the shifter to shift the $A_l \times B_l$ product 32 bits to the right in order to adjust for ranking in the next multiplication-addition sequence. The least significant half of the shift result is stored in the lower 32-bit portion of the accumulator; the upper 32 bits contain the zero and fill.

The second instruction produces the second product, $A_l \times B_m$, adds it to the contents of the accumulator, and stores the result in the accumulator for use in the third instruction.

Instruction 3 computes $A_m \times B_l$, adds the result to the accumulator, and outputs the least significant 32 bits of the addition for use as bits 63-32 of the final product.

This instruction also shifts the result 32 bits to the right to provide the necessary rank adjustment and stores the shift result (the most significant half of the addition result) in the lower 32 bits of the accumulator. Bits ACC63-ACC32 are filled with zeros; the sign is extended into the three upper bits (ACC66-ACC64).

Instruction 4 computes the fourth product ($A_m \times B_m$), adds it to the accumulator, and outputs the least significant half at the Y port for use as bits 95-64 of the final product.

This example assumes that the chip is operating in feed-through mode. A fifth instruction is therefore required to perform the fourth iteration again so that bits 127-96 of the final product can be output.



Example 1. Single-Precision Multiply, 32-Bit Result

| Operand Select R bus S bus EA EB | Instruction Inputs | | | | | | D-MUX Select SELD | Sign Extend SGNEXT | Shift-MUX Control SFT1 SFT0 | | Register Load Select SELREG | Register Write Enable WEH WEL | Feed- through Control FT1 FT0 | | Clock Enables | | | | Y-MUX Select SELY | Y/PY Output Enable OEY | | |
|--|--------------------|-----|----------------------------------|-----|-------------------------------------|---|-------------------------|--------------------------|-----------------------------------|-----|--------------------------------------|--|--|-----|---------------|---|---|---|-------------------------|---------------------------------|--|--|
| | Sign | | Rounding Control RND1 RND0 | | Product Comple- ment COMPL | Multiplier/ Adder Mode ACC1 ACC0 | | | | | | | | | I | A | B | Y | | | | |
| | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | |
| 1 1 | 1 1 | 0 1 | 0 | 0 0 | 0 | X | 0 0 | 0 | 0 0 | 0 0 | 0 0 | 0 0 | 0 0 | 0 0 | 1 | 0 | | | | | | |

Example 2. Double-Precision Multiply, 64-Bit Result

| Instruction Number | Operand Select R bus S bus EA EB | Instruction Inputs | | | | | | D-MUX Select SELD | Sign Extend SGNEXT | Shift-MUX Control SFT1 SFT0 | | Register Load Select SELREG | Register Write Enable WEH WEL | | Feed- through Control FT1 FT0 | | Clock Enables | | | | Y-MUX Select SELY | Y/PY Output Enable OEY |
|-----------------------|--|---------------------|-----|----------------------------------|-----|-------------------------------------|---|-------------------------|--------------------------|-----------------------------------|---|--------------------------------------|--|-----|--|-----|---------------|---|---|---|-------------------------|---------------------------------|
| | | Sign DASGN DBSGN | | Rounding Control RND1 RND0 | | Product Comple- ment COMPL | Multiplier/ Adder Mode ACC1 ACC0 | | | | | | | | | | I | A | B | Y | | |
| | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | |
| (1) | 1 1 | 0 0 | 0 0 | 0 | 0 0 | 0 | 0 0 | 0 | 0 | 1 1 | 0 | 0 0 | 0 0 | 0 0 | 1 1 | 1 1 | 0 | 0 | | | | |
| (2) | 1 1 | 0 1 | 0 0 | 0 | 0 1 | 0 | 0 1 | 0 | 0 | 0 0 | 0 | 0 0 | 0 0 | 0 0 | 1 1 | 1 1 | X | X | | | | |
| (3) | 1 1 | 1 0 | 0 0 | 0 | 0 1 | 0 | 0 1 | 0 | 0 | 1 1 | 0 | 0 0 | 0 0 | 0 0 | 1 1 | 1 1 | 0 | 0 | | | | |
| (4) | 1 1 | 1 1 | 0 0 | 0 | 0 1 | X | 0 1 | X | 0 | 0 0 | X | X X | 0 0 | 0 0 | 1 1 | 1 1 | 0 | 0 | | | | |
| (5) | 1 1 | 1 1 | 0 0 | 0 | 0 1 | X | 0 1 | X | 0 | 0 0 | X | X X | 0 0 | 0 0 | 1 1 | 1 1 | 1 | 0 | | | | |

Example 3. Newton-Raphson Division

| Instruction Number | Operand Select R bus S bus | Instruction Inputs | | | | | | D-MUX Select | Sign Extend | Shift-MUX Control | | Register Load Select | Register Write Enable | Feed- through Control | | Clock Enables | | | | Y-MUX Select | Y/PY Output Enable | | | |
|-----------------------|-------------------------------------|--------------------|-------|---------------------|------|----------------------------|------------------------------|-----------------|----------------|----------------------|--------|----------------------------|-----------------------------|-----------------------------|-----|---------------|-----|-----|------|-----------------|--------------------------|------|------|-----|
| | | Sign | | Rounding Control | | Product Comple- ment | Multiplier/ Adder Mode | | | | | | | | | I | A | B | Y | | | | | |
| | EA | EB | DASGN | DBSGN | RND1 | RND0 | COMPL | ACC1 | ACC0 | SELD | SGNEXT | SFT1 | SFT0 | SELREG | WEH | WEL | FT1 | FT0 | CKEI | CKEA | CKEB | CKEY | SELY | OEY |
| | Repeat N Times* | | | | | | | | | | | | | | | | | | | | | | | |
| (1) | 0 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| (2) | 0 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| End Repeat | | | | | | | | | | | | | | | | | | | | | | | | |
| (3) | 0 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| (4) | 0 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |

*N = $\frac{32}{2m+1}$ Where m = number of bits in the seed (assuming 32-bits of precision)

SN74ACT8836A 32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

Newton-Raphson binary division algorithm

The following explanation illustrates how to implement the Newton-Raphson binary division algorithm using the 'ACT8836A multiplier/accumulator. The Newton-Raphson algorithm is an iterative procedure that generates the reciprocal of the divisor through a convergence method.

Consider the equation $Q = A/B$. This equation can be rewritten as $Q = A \times (1/B)$. Therefore, the quotient Q can be computed by simply multiplying the dividend A by the reciprocal of the divisor (B). Finding the divisor reciprocal $1/B$ is the objective of the Newton-Raphson algorithm.

To calculate $1/B$ the Newton-Raphson equation, $X_{i+1} = X_i(2-BX_i)$ is calculated in an iterative process. In the equation, B represents the divisor and X represents successively closer approximations to the reciprocal $1/B$. The following sequence of computation illustrates the iterative nature of the Newton-Raphson algorithm.

Step 1 $X_1 = X_0(2-BX_0)$
Step 2 $X_2 = X_1(2-BX_1)$
Step 3 $X_3 = X_2(2-BX_2)$
Step n $X_n = X_{n-1}(2-BX_{n-1})$

The successive approximation of X_i , for all i , approaches the reciprocal $1/B$ as the number of iterations increases; that is

$$\lim_{i \rightarrow n} X_i = 1/B$$

The iterative operation is executed until the desired tolerance or error is reached. The required accuracy for $1/B$ can be determined by subtracting each x_i from its corresponding x_{i+1} . If the difference $|X_{i+1} - X_i|$ is less than or equal to a predetermined round off error, then the process is terminated. The desired tolerance can also be achieved by executing a fixed number of iterations based on the accuracy of the initial guess of $1/B$ stored in RAM or PROM.

The initial guess, X_0 , is called the seed approximation. The seed must be supplied to the Newton-Raphson process externally and must fall within the range of $0 < X_0 < 2/B$ if B is greater than 0 or $2/B < X_0 < 0$ if B is less than 0.

To perform the Newton-Raphson binary division algorithm using the 'ACT8836A, the divisor, B , must be a positive fraction. As a positive fraction, B is limited within the range of $1/2 \leq B < 1$.

Since X_i from Newton-Raphson must lie between $0 < X_i < 2/B$ and since the range of the positive fraction B is $1/2 \leq B < 1$, then the limits of X_i become $1 \leq X_i < 2$.

The range of $-BX_i$ will therefore be $-2 \leq -BX_i \leq -1/2$.

The limits of $-BX_i$ are shown in Table 3 as they would appear in the 'ACT8836A extended bit, binary fraction format.

TABLE 3. LIMITS OF $-BX_i$ IN 'ACT8836A EXTENDED BIT FORMAT

| | Extended Bits | | | 63 | 62 | 61 | | 2 | 1 | 0 |
|------|---------------|----|----|----|----|----|-------|---|---|---|
| | 66 | 65 | 64 | | | | | | | |
| -2 | 1 | 1 | 1 | 0 | 0 | 0 | | 0 | 0 | 0 |
| -1/2 | 1 | 1 | 1 | 1 | 1 | 0 | | 0 | 0 | 0 |

The diagram indicates that $-BX_i$ is always of the form:

1 1 1 d0 . d1 d2.....dn-2 dn-1

The next step in Newton-Raphson is to complete the $2 - BX_i$ equation. The fractional representation of 2 is:

0 0 1 0 . 0 0 0 0

Completion of the $2 - BX_i$ equation is shown in Table 4.

TABLE 4. COMPLETION OF $2 - BX_i$ EQUATION

| Extended Bits | | | 63 | 62 | 61 | | 1 | 0 |
|---------------|----|----|-------|-------|-------|-------|-----------|-----------|
| 66 | 65 | 64 | | | | | | |
| 1 | 1 | 1 | d_0 | d_1 | d_2 | | d_{n-2} | d_{n-1} |
| + | 0 | 0 | 0 | 0 | 0 | | 0 | 0 |
| = | 0 | 0 | d_0 | d_1 | d_2 | | d_{n-2} | d_{n-1} |

Since this step only affects the extended bits (66-64) on the 'ACT8836A, this step can be skipped. The following algorithm can therefore be used to perform Newton-Raphson binary division with the 'ACT8836A.

Assuming B is on the DB bus (or stored in the B register) and X_i is stored in the temporary register:

Step 1

Accumulator $\leftarrow -(\text{DB} \times \text{temporary register})$
 $= 2 - BX_i$

Step 2

Temporary Register \leftarrow Left shift one bit of
 (accumulator times temporary register)
 $= X_{i+1}$
 $= X_i (2 - BX_i)$

Step 3

Repeat Steps 1 and 2 until $|X_{i+1} - X_i| \leq$ a predetermined round-off error

Two cycles are required for each iteration. The left shift that is performed in Step 2 is required to realign X_i after the signed fraction multiply. Microcode for this example is shown in Examples 1, 2, and 3.

SN74ACT8836A

32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

absolute maximum ratings over operating free-air temperature range (unless otherwise noted)[†]

| | |
|--|----------------|
| Supply voltage, V_{CC} (see Note 1) | −0.5 V to 6 V |
| Input clamp current, I_{IK} ($V_I < 0$ or $V_I > V_{CC}$) | ±20 mA |
| Output clamp current, I_{OK} ($V_O < 0$ or $V_O > V_{CC}$) | ±50 mA |
| Continuous output current, I_O ($V_O = 0$ to V_{CC}) | ±50 mA |
| Continuous current through V_{CC} or GND pins | ±100 mA |
| Operating free-air temperature range | 0°C to 70°C |
| Storage temperature range | −65°C to 150°C |

[†]Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTE 1: All voltage levels are with respect to GND.

recommended operating conditions

| | MIN | NOM | MAX | UNIT |
|--|------|-----|----------|------|
| V_{CC} Supply voltage | 4.75 | 5 | 5.25 | V |
| V_{IH} High-level input voltage | 2 | | V_{CC} | V |
| V_{IL} Low-level input voltage | 0 | | 0.8 | V |
| I_{OH} High-level output current | | | −8 | mA |
| I_{OL} Low-level output current | | | 8 | mA |
| V_I Input voltage | 0 | | V_{CC} | V |
| V_O Output voltage | 0 | | V_{CC} | V |
| dt/dv Input transition rise or fall rate | 0 | | 15 | ns/V |
| T_A Operating free-air temperature | 0 | | 70 | °C |

electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

| PARAMETER | TEST CONDITIONS | V_{CC} | MIN | TYP [‡] | MAX | UNIT |
|---------------------------|--|----------|------|------------------|------|------|
| V_{OH} | $I_{OH} = -20 \mu A$ | 4.75 V | 4.6 | | | V |
| | | 5.25 V | 5.1 | | | |
| | $I_{OH} = -8 \text{ mA}$ | 4.75 V | 3.85 | 3.95 | | |
| | | 5.25 V | 4.60 | 4.70 | | |
| V_{OL} | $I_{OL} = 20 \mu A$ | 4.75 V | | | 0.1 | V |
| | | 5.25 V | | | 0.1 | |
| | $I_{OL} = 8 \text{ mA}$ | 4.75 V | | 0.32 | 0.45 | |
| | | 5.25 V | | 0.32 | 0.45 | |
| I_I^{\S} | $V_I = V_{CC}$ or 0 | 5.25 V | | ±0.1 | ±5 | μA |
| I_{CCQ} | $V_I = V_{CC}$ or 0, I_O | 5.25 V | | 100 | 200 | μA |
| C_I | $V_I = V_{CC}$ or 0 | 5 V | | 10 | | pF |
| ΔI_{CC}^{\dagger} | One input at 3.4 V, other inputs at 0 or V_{CC} | 5.25 V | | | 1 | mA |

[‡]All typical values are at $V_{CC} = 5 \text{ V}$, $T_A = 25^\circ\text{C}$.

^{\S}For I/O ports, the parameter I_{OZH} and I_{OZL} include the offstate output current.

^{\dagger}This is the increase in supply current for each input that is at one of the specified TTL voltage levels rather than 0 or V_{CC} .

SN74ACT8836A
32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

timing requirements over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

setup and hold times

| | PARAMETER | MIN | MAX | UNIT |
|-------------------|---|-----|-----|------|
| t _{su1} | Setup time, DA, DB, or I before CLK↑† | 10 | | ns |
| t _{su2} | Setup time, DA, DB, or I before CLK↑, 00 MODE† | 45 | | |
| t _{su3} | Setup time, $\overline{\text{CKEA}}$ before CLK↑ | 10 | | |
| t _{su4} | Setup time, $\overline{\text{CKEB}}$ before CLK↑ | 10 | | |
| t _{su5} | Setup time, $\overline{\text{CKEI}}$ before CLK↑ | 10 | | |
| t _{su6} | Setup time, $\overline{\text{CKEY}}$ before CLK↑ | 10 | | |
| t _{su7} | Setup time, SELREG before CLK↑ | 10 | | |
| t _{su8} | Setup time, $\overline{\text{WEMS}}$ before CLK↑ | 10 | | |
| t _{su9} | Setup time, $\overline{\text{WELS}}$ before CLK↑ | 10 | | |
| t _{su10} | Setup time, SELD before CLK↑ | 10 | | |
| t _{su11} | Setup time, $\overline{\text{EA}}$ or $\overline{\text{EB}}$ before CLK↑, 11 MODE | 26 | | |
| t _{su12} | Setup time, $\overline{\text{EA}}$ or $\overline{\text{EB}}$ before CLK↑, 10 MODE | 45 | | |
| t _{su13} | Setup time, $\overline{\text{EA}}$ or $\overline{\text{EB}}$ before CLK↑, 01 MODE | 42 | | |
| t _{su14} | Setup time, $\overline{\text{EA}}$ or $\overline{\text{EB}}$ before CLK↑, 00 MODE | 45 | | |
| t _{h1} | Hold time, DA, DB, or I after CLK↑† | 0 | | ns |
| t _{h2} | Hold time, DA, DB, or I after CLK↑, 00 MODE† | 0 | | |
| t _{h3} | Hold time, $\overline{\text{CKEA}}$ after CLK↑ | 0 | | |
| t _{h4} | Hold time, $\overline{\text{CKEB}}$ after CLK↑ | 0 | | |
| t _{h5} | Hold time, $\overline{\text{CKEI}}$ after CLK↑ | 0 | | |
| t _{h6} | Hold time, $\overline{\text{CKEY}}$ after CLK↑ | 0 | | |
| t _{h7} | Hold time, SELREG after CLK↑ | 0 | | |
| t _{h8} | Hold time, $\overline{\text{WEMS}}$ after CLK↑ | 0 | | |
| t _{h9} | Hold time, $\overline{\text{WELS}}$ after CLK↑ | 0 | | |
| t _{h10} | Hold time, SELD after CLK↑ | 0 | | |
| t _{h11} | Hold time, $\overline{\text{EA}}$ or $\overline{\text{EB}}$ after CLK↑ | 0 | | |
| t _{h12} | Hold time, $\overline{\text{EA}}$ or $\overline{\text{EB}}$ after CLK↑ | 0 | | |
| t _{h13} | Hold time, $\overline{\text{EA}}$ or $\overline{\text{EB}}$ after CLK↑ | 0 | | |
| t _{h14} | Hold time, $\overline{\text{EA}}$ or $\overline{\text{EB}}$ after CLK↑ | 0 | | |

†The notation "I" includes all inputs to the Instruction register. These are DASGN, DBSGN, RND1-RND0, COMPL, and ACC1-ACC0.

cycle times

| | FT MODE | MIN (ns) | COMMENTS |
|----|---------------------|----------|---|
| 00 | Without Accumulator | 48 | Equal to t _{pd10} , see Note 2 |
| 00 | With Accumulator | 45 | Equal to t _{su2} , see Notes 2 and 3 |
| 01 | Without Accumulator | 45 | see Note 3 |
| 01 | With Accumulator | 45 | see Note 3 |
| 10 | Without Accumulator | 48 | see Note 3 |
| 10 | With Accumulator | 48 | see Note 3 |
| 11 | Without Accumulator | 30 | see Note 3 |
| 11 | With Accumulator | 30 | see Note 3 |

NOTES: 2. The FTA mode must include t_{su2} because the set-up time is longer than the CLK to Y propagation delay.

3. All cycle times should include t_{pd13} if all 64 bits of result are to be obtained in a single cycle.

clock requirements

| | PARAMETER | MIN | MAX | UNIT |
|----------------|-----------|-----|-----|------|
| t _w | CLK high | 10 | | ns |
| | CLK low | 15 | | |



SN74ACT8836A **32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR**

switching characteristics over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)[†]

| PARAMETER | FROM (INPUT) | TO (OUTPUT) | FT MODE | MIN | MAX | UNIT |
|-------------------|----------------|-------------|-----------|-----|-----|------|
| t _{pd1} | Y Reg | Y | 01 or 11 | | 15 | ns |
| t _{pd2} | A, B, or I Reg | | 10 | | 45 | ns |
| t _{pd3} | Accumulator | | 00 | | 38 | ns |
| t _{pd4} | Y Reg | YETP | 01 or 11 | | 15 | ns |
| t _{pd5} | A, B, or I Reg | | 10 | | 45 | ns |
| t _{pd6} | Accumulator | | 00 | | 38 | ns |
| t _{pd7} | Y Reg | ETPERR | 01 or 11 | | 15 | ns |
| t _{pd8} | A, B, or I Reg | | 10 | | 45 | ns |
| t _{pd9} | Accumulator | | 00 | | 38 | ns |
| t _{pd10} | DATA, I | Y | 00 | | 48 | ns |
| t _{pd11} | | YETP | 00 | | 48 | ns |
| t _{pd12} | | ETPERR | 00 | | 48 | ns |
| t _{pd13} | SELY | Y | All modes | | 12 | ns |
| t _{pd14} | DA | PERRA | All modes | | 16 | ns |
| t _{pd15} | PA | | All modes | | 16 | ns |
| t _{pd16} | DB | PERRB | All modes | | 16 | ns |
| t _{pd17} | PB | | All modes | | 16 | ns |
| t _{pd18} | Y | MSERR | All modes | | 16 | ns |
| t _{pd19} | PY | PERRY | All modes | | 16 | ns |
| t _{pd20} | YETP | MSERR | All modes | | 16 | ns |
| t _{pd21} | TPO-1 | All outputs | All modes | | 48 | ns |
| t _{en} | OEY | Y, YETP | All modes | | 15 | ns |
| t _{dis} | | | All modes | | 13 | ns |

[†]See Parameter Measurement Information for load circuits and voltage waveforms.

| TT MODE | MIN (ns) | COMMENT |
|---------|----------|---------------------|
| 00 | 40 | Without Accumulator |
| 01 | 40 | With Accumulator |
| 02 | 40 | Without Accumulator |
| 03 | 40 | With Accumulator |
| 04 | 40 | Without Accumulator |
| 05 | 40 | With Accumulator |
| 06 | 40 | Without Accumulator |
| 07 | 40 | With Accumulator |
| 08 | 40 | Without Accumulator |
| 09 | 40 | With Accumulator |
| 10 | 40 | Without Accumulator |
| 11 | 40 | With Accumulator |

| PARAMETER | MIN | MAX | UNIT |
|------------------------|-----|-----|------|
| t _{CL} (high) | 10 | | ns |
| t _{CL} (low) | 10 | | ns |



PARAMETER MEASUREMENT INFORMATION

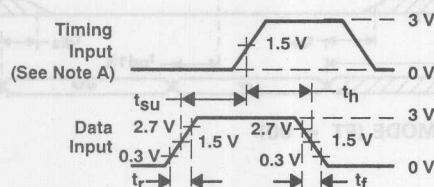
LOAD CIRCUIT PARAMETERS

| TIMING PARAMETERS | C_{LOAD}^{\dagger} (pF) | I_{OL} (mA) | I_{OH} (mA) | V_{LOAD} (V) |
|----------------------|------------------------------|------------------|------------------|-------------------|
| t_{en} | 50 | 8 | -8 | 0 |
| t_{pZH} | | | | 3 |
| t_{pZL} | 50 | 8 | -8 | 1.5 |
| t_{dis} | | | | 1.5 |
| t_{pd} | 50 | 8 | -8 | ‡ |

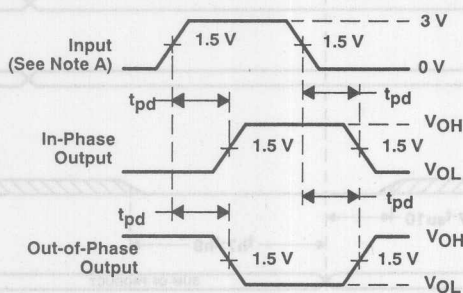
† C_{LOAD} includes probes and test fixture capacitance.

‡ $V_{LOAD} - V_{OL} = 50 \Omega$, where $V_{OL} = 0.6 V$, $I_{OL} = 8 mA$.

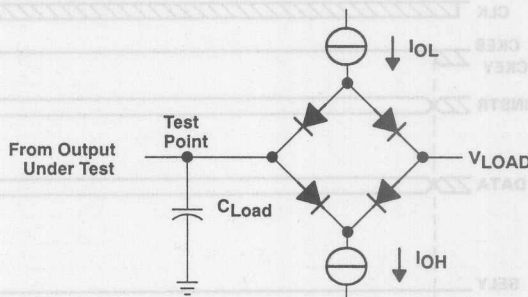
I_{OL}



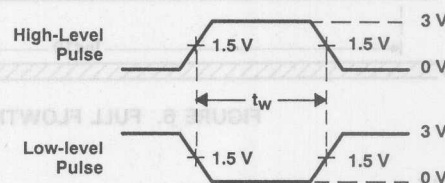
VOLTAGE WAVEFORMS
SETUP AND HOLD TIMES
INPUT RISE AND FALL TIMES



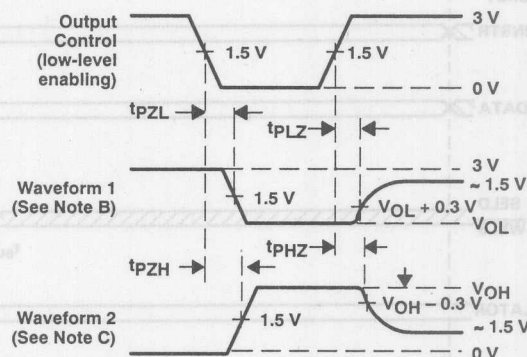
VOLTAGE WAVEFORMS
PROPAGATION DELAY TIMES



LOAD CIRCUIT



VOLTAGE WAVEFORMS
PULSE DURATION



VOLTAGE WAVEFORMS
ENABLE AND DISABLE TIMES, 3-STATE OUTPUTS

- Notes: A. Phase relationships between waveforms were chosen arbitrarily. All input pulses are supplied by pulse generators having the following characteristics: PRR = 1 MHz, $Z_O = 50 \Omega$, $t_r \leq 6 ns$, $t_f \leq 6 ns$.
B. Waveform 1 is for an output with internal conditions such that the output is low except when disabled by the output control.
C. Waveform 2 is for an output with internal conditions such that the output is high except when disabled by the output control. For t_{PLZ} and t_{PHZ} , V_{OL} and V_{OH} are specified values.

FIGURE 5

SN74ACT8836A
32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

PARAMETER MEASUREMENT INFORMATION

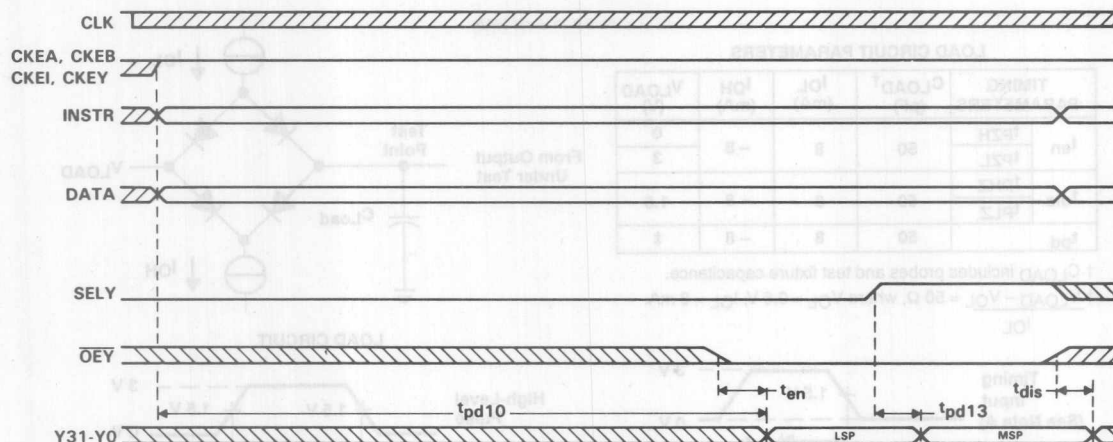


FIGURE 6. FULL FLOWTHROUGH MODE (FT = 00)

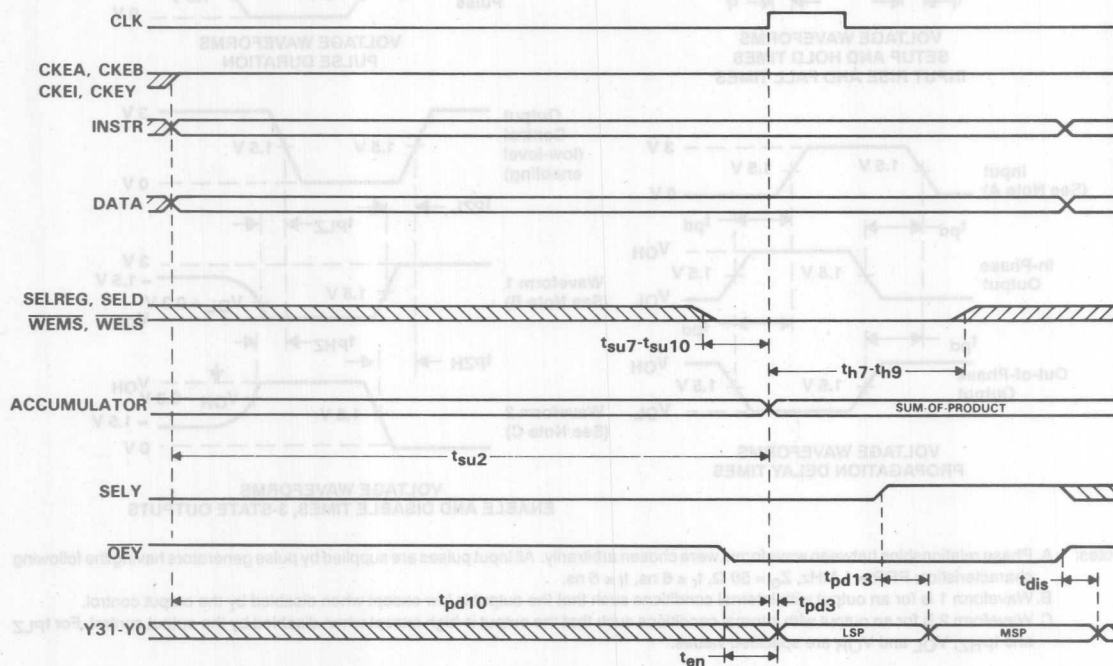


FIGURE 7. FULL FLOWTHROUGH MODE, ACCUMULATOR MODE (FT = 00)

PARAMETER MEASUREMENT INFORMATION

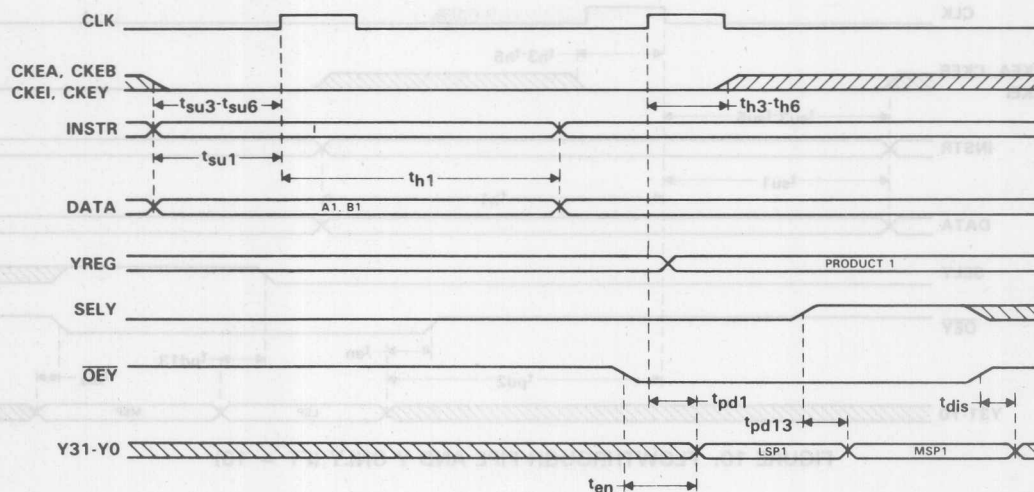


FIGURE 8. FLOWTHROUGH PIPE ONLY (FT = 01)

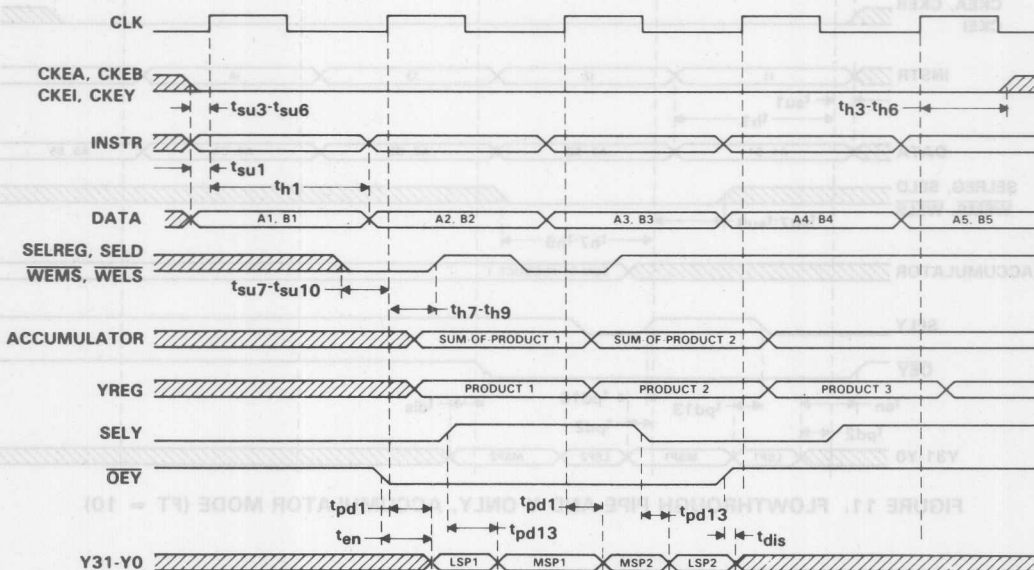


FIGURE 9. FLOWTHROUGH PIPE ONLY, ACCUMULATION MODE (FT = 01)

SN74ACT8836A
32-BIT BY 32-BIT MULTIPLIER/ACCUMULATOR

PARAMETER MEASUREMENT INFORMATION

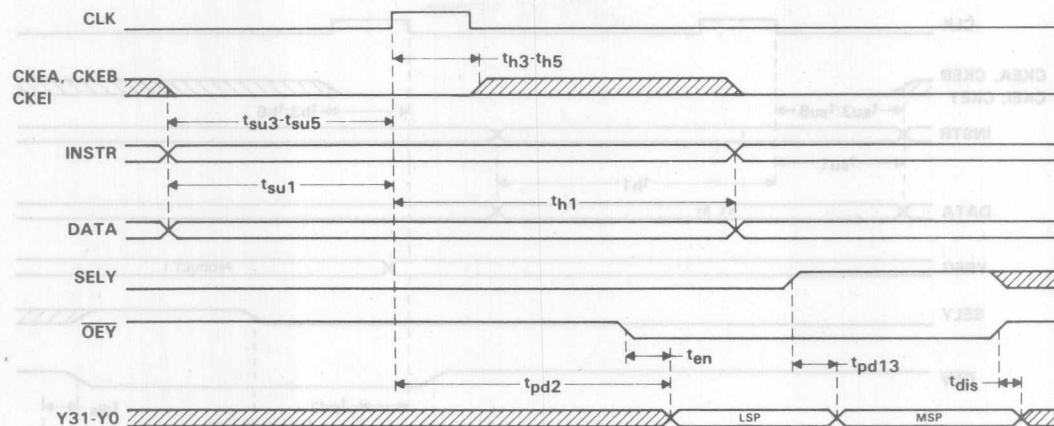


FIGURE 10. FLOWTHROUGH PIPE AND Y ONLY (FT = 10)

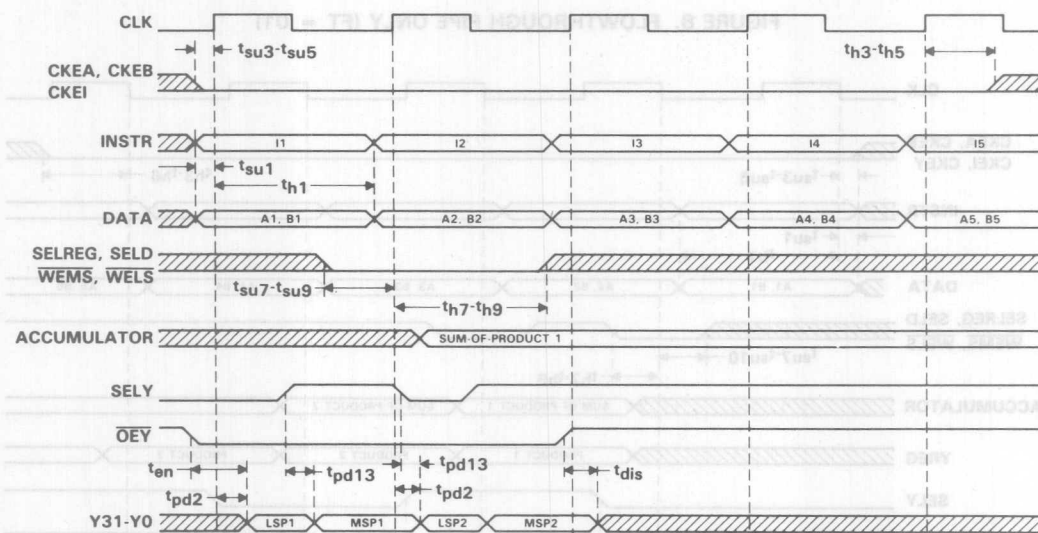


FIGURE 11. FLOWTHROUGH PIPE AND Y ONLY, ACCUMULATOR MODE (FT = 10)

PARAMETER MEASUREMENT INFORMATION

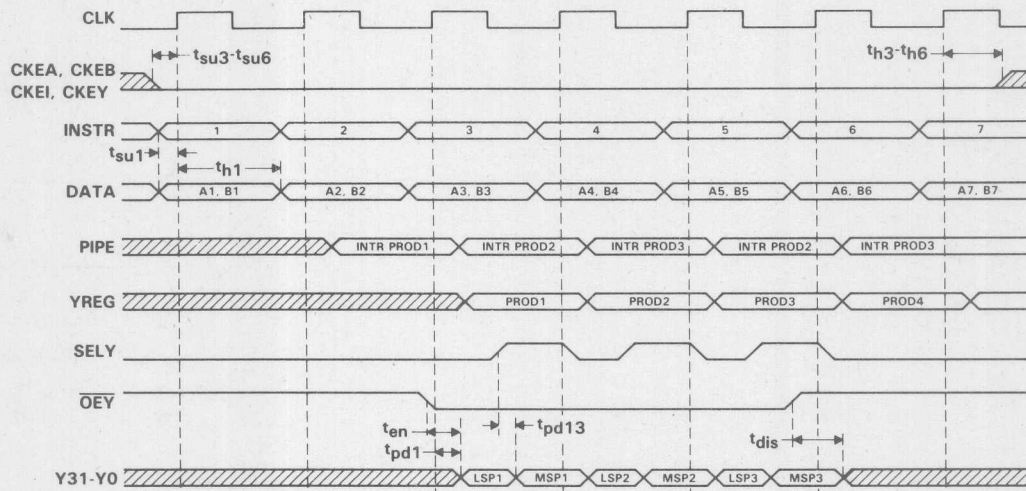


FIGURE 12. ALL REGISTERS ENABLED (FT = 11)

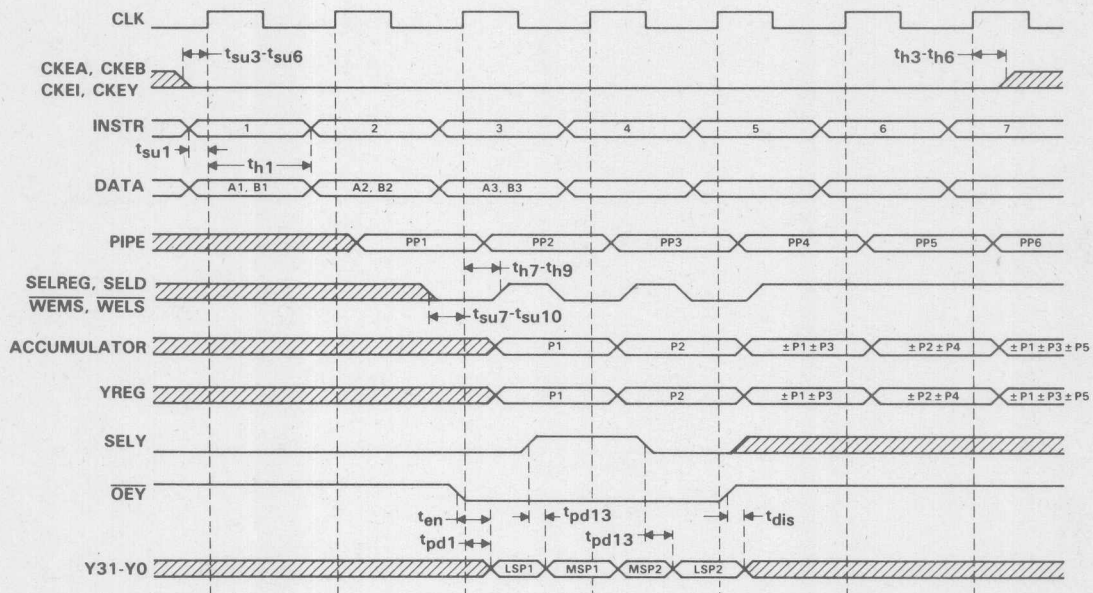


FIGURE 13. ALL REGISTERS ENABLED, ACCUMULATOR MODE (FT = 11)

PARAMETER MEASUREMENT INFORMATION

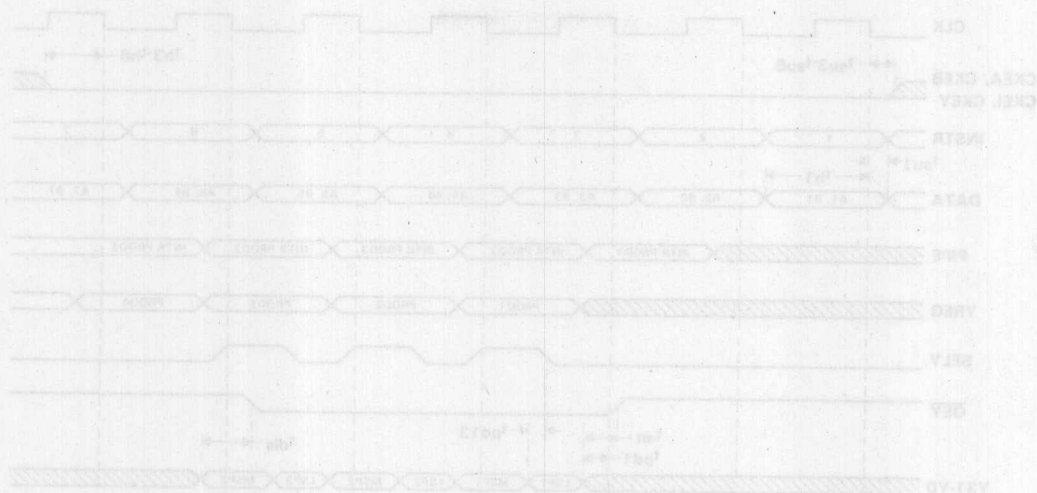


FIGURE 12. ALL REGISTERS ENABLED (FT = 1)

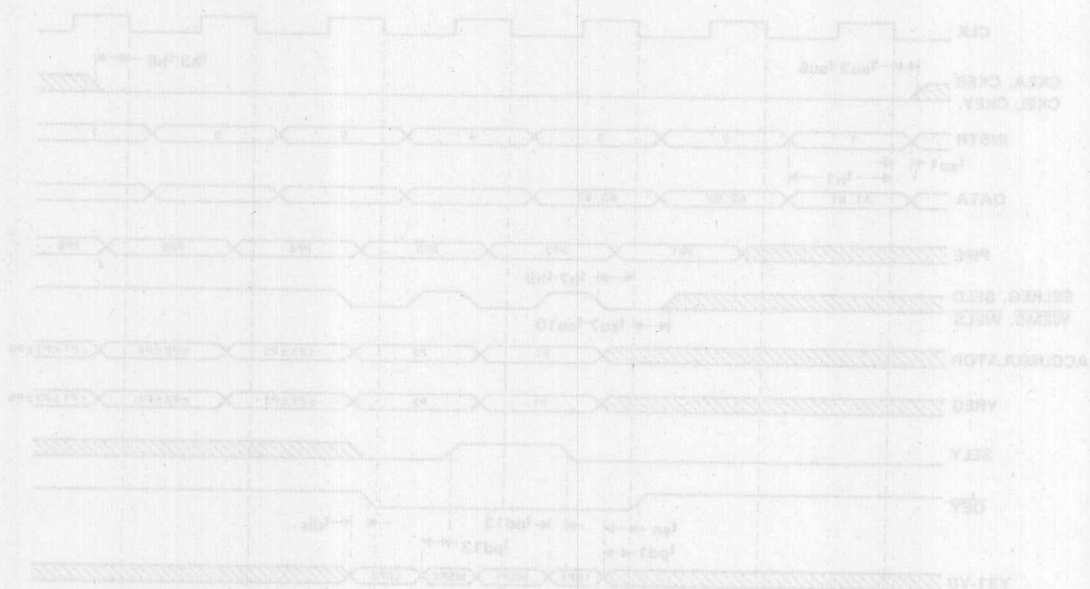


FIGURE 13. ALL REGISTERS ENABLED, ACCUMULATOR MODE (FT = 1)

| | |
|--|----|
| General Information | 1 |
| SN74ACT8818A 16-Bit Microsequencer | 2 |
| SN74ACT8832A 32-Bit Registered ALU | 3 |
| SN74ACT8836A 32- × 32-Bit Parallel Multiplier | 4 |
| SN74ACT8841A Digital Crossbar Switch | 5 |
| SN74ACT8847 64-Bit Floating-Point/Integer Unit | 6 |
| SN74ACT8847 Application Information | 7 |
| SN74ACT8867 32-Bit Vector Processor Unit | 8 |
| Design Support | 9 |
| Mechanical Data | 10 |

1

General Information

2

SWTACT8841A 16-Bit Microprocessor

3

SWTACT8833A 32-Bit Microprocessor

4

SWTACT8833A 32 x 32-Bit Parallel Multiplier

5

SWTACT8841A Digital Crossover Switch

6

SWTACT8837 64-Bit Floating-Point Special Unit

7

SWTACT8863 Application Processor

8

SWTACT8837 32-Bit Vector Processor Unit

9

Design Support

10

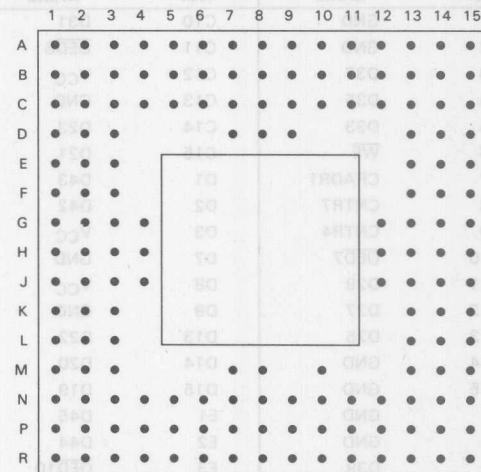
Mechanical Data

SN74ACT8841A DIGITAL CROSSBAR SWITCH

D3400, JANUARY 1990

- High-Speed Programmable Switch for Parallel Processing Applications
- Dynamically Reconfigurable for Fault-Tolerant Routing
- 64 Bidirectional Data I/Os in 16 Nibble Groups of Four Bits Each
- Data I/O Selection Programmable by Nibble
- Eight Banks of Control Flip-Flops for Storing Configuration Programs
- Two Selectable Hard-Wired Switching Configurations
- Selectable Stored-Data or Real-Time Inputs
- 156-Pin Grid-Array Package
- Low-Power 1- μ m EPIC™ CMOS Process
- Single 5-V Power Supply

15 × 15 GC PACKAGE
(TOP VIEW)



description

The SN74ACT8841A is a flexible, high-speed digital crossbar switch. It is easily microprogrammable to support user-definable interconnection patterns. This crossbar switch is especially suited to multiprocessor interconnects that are dynamically reconfigurable or even reprogrammable after each system clock. The 'ACT8841A is built in Texas Instruments advanced 1- μ m EPIC™ CMOS process to enhance performance and reduce power consumption. The switch requires only a 5-V power supply.

Because the 'ACT8841A is a 16-port device, system architectures based on the 'ACT8841A can include up to 16 switching nodes, which may be processors, data memories, or bus interfaces. Larger processor arrays can be built with multistage interconnection schemes. Most applications will use the crossbar switch as a broadband bus interface controller, for example, between closely coupled processors that must exchange data with very low propagation delays.

The 'ACT8841A has ten selectable control sources, including eight banks of programmable control flip-flops and two hard-wired control circuits. The device can switch from 1 to 16 nibbles (4 to 64 bits) of data in a single cycle.

The 64 I/O pins of the 'ACT8841A are arranged in 16 switchable nibbles (see Figure 1). A single input nibble can be broadcast to any combination of 15 output nibbles, or even to 16 nibbles (including itself) if operating off registered data. Multiple input nibbles can be switched to multiple outputs, depending on the programmed configurations available in the control flip-flops.

The digital crossbar switch is intended primarily for multiprocessor interconnection and parallel processing applications. The device can be used to select and transfer data from multiple sources to multiple destinations. Since it can be dynamically reprogrammed, it is suitable for use in reconfigurable networks for fault-tolerant routing.

The 'ACT8841A is characterized for operation from 0°C to 70°C.

EPIC is a trademark of Texas Instruments Incorporated.

PRODUCTION DATA documents contain information current as of publication date. Products conform to specifications per the terms of Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.

TEXAS
INSTRUMENTS

Copyright © 1990, Texas Instruments Incorporated

POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

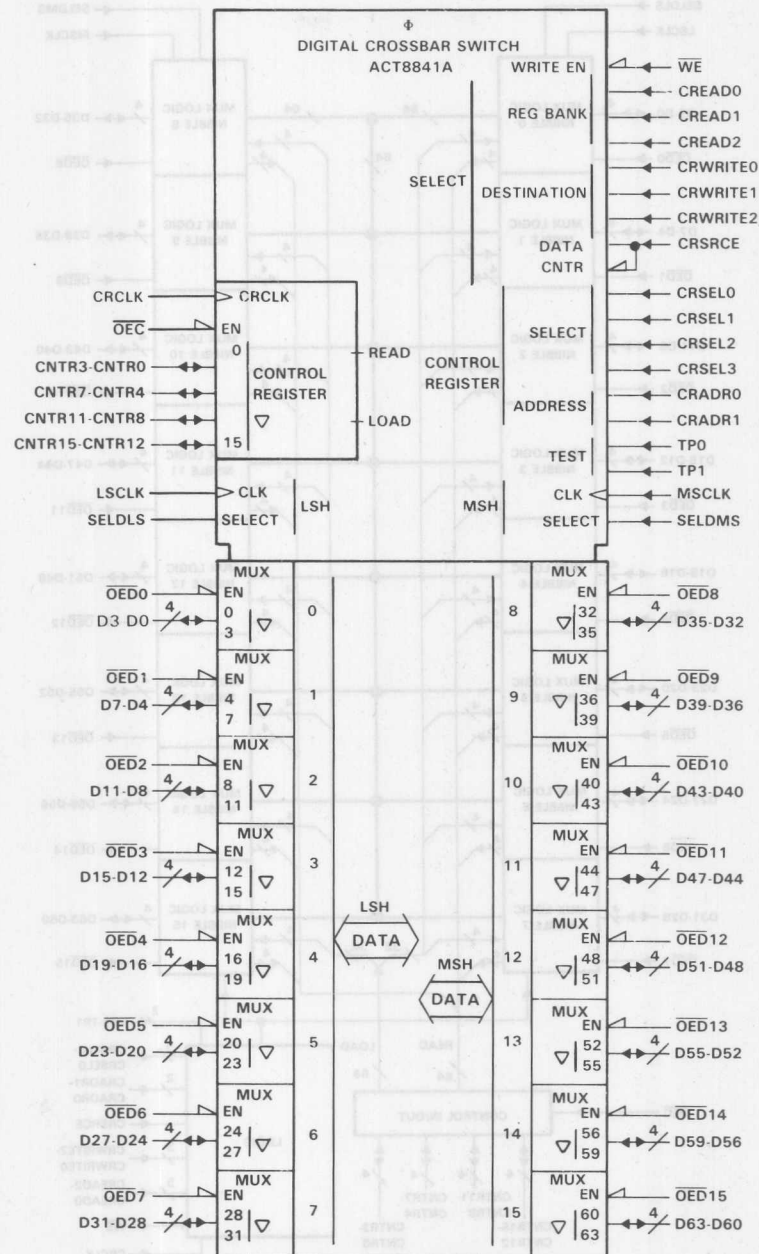
SN74ACT8841A **DIGITAL CROSSBAR SWITCH**

PIN ASSIGNMENTS

| NO. | NAME | NO. | NAME | NO. | NAME | NO. | NAME |
|-----|--------------------------|-----|---------------------------|-----|---------------------------|-----|---------------------------|
| A1 | GND | C10 | D31 | H12 | VCC | N7 | CNTR13 |
| A2 | GND | C11 | $\overline{\text{OED6}}$ | H13 | LSCLK | N8 | CREADO |
| A3 | D37 | C12 | VCC | H14 | SELDLS | N9 | VCC |
| A4 | D35 | C13 | GND | H15 | CNTR3 | N10 | D0 |
| A5 | D33 | C14 | D23 | J1 | $\overline{\text{OEC}}$ | N11 | D3 |
| A6 | $\overline{\text{WE}}$ | C15 | D21 | J2 | CRWRITE0 | N12 | D6 |
| A7 | CRADR1 | D1 | D43 | J3 | CRWRITE1 | N13 | GND |
| A8 | CNTR7 | D2 | D42 | J4 | GND | N14 | D8 |
| A9 | CNTR4 | D3 | VCC | J12 | GND | N15 | D9 |
| A10 | $\overline{\text{OED7}}$ | D7 | GND | J13 | CNTR2 | P1 | GND |
| A11 | D29 | D8 | VCC | J14 | CNTR1 | P2 | GND |
| A12 | D27 | D9 | GND | J15 | CNTR0 | P3 | D56 |
| A13 | D25 | D13 | D22 | K1 | CRWRITE2 | P4 | D58 |
| A14 | GND | D14 | D20 | K2 | $\overline{\text{OED12}}$ | P5 | D60 |
| A15 | GND | D15 | D19 | K3 | D48 | P6 | D62 |
| B1 | GND | E1 | D45 | K13 | D15 | P7 | CNTR12 |
| B2 | GND | E2 | D44 | K14 | D14 | P8 | CNTR15 |
| B3 | D39 | E3 | $\overline{\text{OED10}}$ | K15 | $\overline{\text{OED3}}$ | P9 | TPO |
| B4 | D36 | E13 | $\overline{\text{OED5}}$ | L1 | D49 | P10 | $\overline{\text{OED0}}$ |
| B5 | D34 | E14 | D18 | L2 | D50 | P11 | D2 |
| B6 | $\overline{\text{OED8}}$ | E15 | D17 | L3 | $\overline{\text{OED13}}$ | P12 | D4 |
| B7 | CRADRO | F1 | $\overline{\text{OED11}}$ | L13 | $\overline{\text{OED2}}$ | P13 | D7 |
| B8 | CRSRCE | F2 | D46 | L14 | D12 | P14 | GND |
| B9 | CNTR5 | F3 | D47 | L15 | D13 | P15 | GND |
| B10 | D30 | F13 | D16 | M1 | D51 | R1 | GND |
| B11 | D28 | F14 | $\overline{\text{OED4}}$ | M2 | D52 | R2 | GND |
| B12 | D26 | F15 | CRSEL3 | M3 | D54 | R3 | D57 |
| B13 | D24 | G1 | CNTR8 | M7 | GND | R4 | D59 |
| B14 | GND | G2 | CNTR9 | M8 | VCC | R5 | D61 |
| B15 | GND | G3 | CNTR10 | M10 | GND | R6 | $\overline{\text{OED15}}$ |
| C1 | D41 | G4 | GND | M13 | VCC | R7 | CNTR14 |
| C2 | D40 | G12 | GND | M14 | D10 | R8 | CREAD1 |
| C3 | GND | G13 | CRSEL2 | M15 | D11 | R9 | CREAD2 |
| C4 | D38 | G14 | CRSEL1 | N1 | D53 | R10 | TP1 |
| C5 | $\overline{\text{OED9}}$ | G15 | CRSELO | N2 | D55 | R11 | D1 |
| C6 | D32 | H1 | CNTR11 | N3 | GND | R12 | $\overline{\text{OED1}}$ |
| C7 | VCC | H2 | SELDMS | N4 | VCC | R13 | D5 |
| C8 | CRCLK | H3 | MSCLK | N5 | $\overline{\text{OED14}}$ | R14 | GND |
| C9 | CNTR6 | H4 | VCC | N6 | D63 | R15 | GND |

SN74ACT8841A DIGITAL CROSSBAR SWITCH

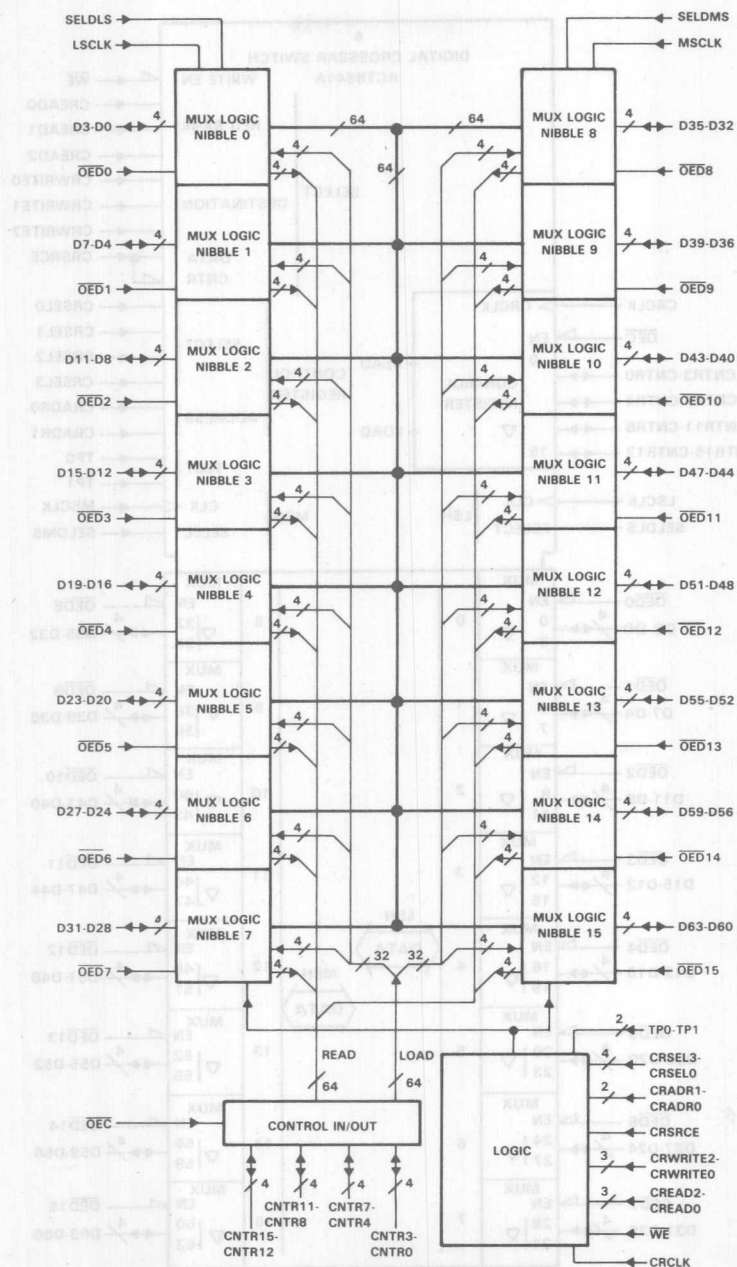
logic symbol†



†This symbol is in accordance with ANSI/IEEE Std 91-1984.

SN74ACT8841A DIGITAL CROSSBAR SWITCH

functional block diagram



TERMINAL FUNCTIONS

| PIN | | I/O | DESCRIPTION | QU | QV | QW |
|----------|-----|-----|--|----|----|----|
| NAME | NO. | | | | | |
| CNTR0 | J15 | I/O | Control I/O. Inputs four control words to the control flip-flops on each CRCLK cycle. As outputs, the same addresses can be used to read the flip-flop settings. | | | |
| CNTR1 | J14 | | | | | |
| CNTR2 | J13 | | | | | |
| CNTR3 | H15 | | | | | |
| CNTR4 | A9 | | | | | |
| CNTR5 | B9 | | | | | |
| CNTR6 | C9 | | | | | |
| CNTR7 | A8 | | | | | |
| CNTR8 | G1 | | | | | |
| CNTR9 | G2 | | | | | |
| CNTR10 | G3 | | | | | |
| CNTR11 | H1 | | | | | |
| CNTR12 | P7 | | | | | |
| CNTR13 | N7 | | | | | |
| CNTR14 | R7 | | | | | |
| CNTR15 | P8 | | | | | |
| CRADRO | B7 | I | Control register address. Selects 16-bits of control flip-flops as a source/destination for outputs/inputs on CNTR0-CNTR15. (see Table 7) | | | |
| CRADR1 | A7 | | | | | |
| CRCLK | C8 | I | Control register clock. Clocks CNTR0-CNTR15 into the control flip-flops on low-to-high transition. | | | |
| CREADO | N8 | I | Selects one of eight banks of control flip-flops to read out on CNTR0-CNTR15 in 16-bit words addressed by CRADR1-CRADRO. | | | |
| CREAD1 | R8 | | | | | |
| CREAD2 | R9 | | | | | |
| CRSEL0 | G15 | I | Selects one of ten control configurations. | | | |
| CRSEL1 | G14 | | | | | |
| CRSEL2 | G13 | | | | | |
| CRSEL3 | F15 | | | | | |
| CRSRCE | B8 | I | Load source select. When low selects CNTR inputs, when high selects DATA inputs. | | | |
| CRWRITE0 | J2 | I | Destination select. Selects one of eight control banks. (see Table 4) | | | |
| CRWRITE1 | J3 | | | | | |
| CRWRITE2 | K1 | | | | | |
| D0 | N10 | I/O | I/O data bits 0 through 17 (data bits 0 through 31 are the least significant half). | | | |
| D1 | R11 | | | | | |
| D2 | P11 | | | | | |
| D3 | N11 | | | | | |
| D4 | P12 | | | | | |
| D5 | R13 | | | | | |
| D6 | N12 | | | | | |
| D7 | P13 | | | | | |
| D8 | N14 | | | | | |
| D9 | N15 | | | | | |
| D10 | M14 | | | | | |
| D11 | M15 | | | | | |
| D12 | L14 | | | | | |
| D13 | L15 | | | | | |
| D14 | K14 | | | | | |
| D15 | K13 | | | | | |
| D16 | F13 | | | | | |
| D17 | E15 | | | | | |

SN74ACT8841A **DIGITAL CROSSBAR SWITCH**

TERMINAL FUNCTIONS (continued)

| PIN | NAME | NO. | I/O | DESCRIPTION | DESCRIPTION | NO. | NAME |
|-----|------|-----|--|-------------|-------------|-----|--------|
| D18 | E14 | | | | | 715 | CHTR0 |
| D19 | D15 | | | | | 714 | CHTR1 |
| D20 | D14 | | | | | 713 | CHTR2 |
| D21 | C15 | | | | | 712 | CHTR3 |
| D22 | D13 | | | | | 711 | CHTR4 |
| D23 | C14 | | | | | 710 | CHTR5 |
| D24 | B13 | | | | | 709 | CHTR6 |
| D25 | A13 | I/O | I/O data bits 18 through 31 (data bits 0 through 31 are the least significant half). | | | 708 | CHTR7 |
| D26 | B12 | | | | | 707 | CHTR8 |
| D27 | A12 | | | | | 706 | CHTR9 |
| D28 | B11 | | | | | 705 | CHTR10 |
| D29 | A11 | | | | | 704 | CHTR11 |
| D30 | B10 | | | | | 703 | CHTR12 |
| D31 | C10 | | | | | 702 | CHTR13 |
| D32 | C6 | | | | | 701 | CHTR14 |
| D33 | A5 | | | | | 700 | CHTR15 |
| D34 | B5 | | | | | 699 | CHTR16 |
| D35 | A4 | | | | | 698 | CHTR17 |
| D36 | B4 | | | | | 697 | CHTR18 |
| D37 | A3 | | | | | 696 | CHTR19 |
| D38 | C4 | | | | | 695 | CHTR20 |
| D39 | B3 | | | | | 694 | CHTR21 |
| D40 | C2 | | | | | 693 | CHTR22 |
| D41 | C1 | | | | | 692 | CHTR23 |
| D42 | D2 | | | | | 691 | CHTR24 |
| D43 | D1 | | | | | 690 | CHTR25 |
| D44 | E2 | | | | | 689 | CHTR26 |
| D45 | E1 | | | | | 688 | CHTR27 |
| D46 | F2 | | | | | 687 | CHTR28 |
| D47 | F3 | | | | | 686 | CHTR29 |
| D48 | K3 | I/O | I/O data bits 32 through 63 (data bits 32 through 63 are the most significant half). | | | 685 | CHTR30 |
| D49 | L1 | | | | | 684 | CHTR31 |
| D50 | L2 | | | | | 683 | CHTR32 |
| D51 | M1 | | | | | 682 | CHTR33 |
| D52 | M2 | | | | | 681 | CHTR34 |
| D53 | N1 | | | | | 680 | CHTR35 |
| D54 | M3 | | | | | 679 | CHTR36 |
| D55 | N2 | | | | | 678 | CHTR37 |
| D56 | P3 | | | | | 677 | CHTR38 |
| D57 | R3 | | | | | 676 | CHTR39 |
| D58 | P4 | | | | | 675 | CHTR40 |
| D59 | R4 | | | | | 674 | CHTR41 |
| D60 | P5 | | | | | 673 | CHTR42 |
| D61 | R5 | | | | | 672 | CHTR43 |
| D62 | P6 | | | | | 671 | CHTR44 |
| D63 | N6 | | | | | 670 | CHTR45 |

SN74ACT8841A DIGITAL CROSSBAR SWITCH

TERMINAL FUNCTIONS (continued)

| PIN NAME | NO. | I/O | DESCRIPTION |
|---------------------------|-----|-----|--|
| GND | A1 | | |
| GND | A2 | | |
| GND | A14 | | |
| GND | A15 | | |
| GND | B1 | | |
| GND | B2 | | |
| GND | B14 | | |
| GND | B15 | | |
| GND | C3 | | |
| GND | C13 | | |
| GND | D7 | | |
| GND | D9 | | |
| GND | G4 | | |
| GND | G12 | | |
| GND | J4 | | |
| GND | J12 | | |
| GND | M7 | | |
| GND | M10 | | |
| GND | N3 | | |
| GND | N13 | | |
| GND | P1 | | |
| GND | P2 | | |
| GND | P14 | | |
| GND | P15 | | |
| GND | R1 | | |
| GND | R2 | | |
| GND | R14 | | |
| GND | R15 | | |
| LSCLK | H13 | I | Clocks the least significant half of data inputs into the input registers on a low-to-high transition. |
| MSCLK | H3 | I | Clocks the most significant half of data inputs into the input registers on a low-to-high transition. |
| OEC | J1 | I | Output enable for control flip-flops, active low |
| $\overline{\text{OED}}0$ | P10 | | |
| $\overline{\text{OED}}1$ | R12 | | |
| $\overline{\text{OED}}2$ | L13 | | |
| $\overline{\text{OED}}3$ | K15 | | |
| $\overline{\text{OED}}4$ | F14 | | |
| $\overline{\text{OED}}5$ | E13 | | |
| $\overline{\text{OED}}6$ | C11 | | |
| $\overline{\text{OED}}7$ | A10 | | |
| $\overline{\text{OED}}8$ | B6 | | |
| $\overline{\text{OED}}9$ | C5 | | |
| $\overline{\text{OED}}10$ | E3 | | |
| $\overline{\text{OED}}11$ | F1 | | |
| $\overline{\text{OED}}12$ | K2 | | |
| $\overline{\text{OED}}13$ | L3 | | |
| $\overline{\text{OED}}14$ | N5 | | |
| $\overline{\text{OED}}15$ | R6 | | |

SN74ACT8841A **DIGITAL CROSSBAR SWITCH**

TERMINAL FUNCTIONS (continued)

| PIN | | I/O | DESCRIPTION | I/O | NAME |
|--------|-----|-----|---|-----|------|
| NAME | NO. | | | | |
| SELDLS | H14 | I | When low, selects the stored, least significant data input to the main internal bus. When high, real-time data is selected. | | |
| SELDMS | H2 | I | When low, selects the stored, most significant data input to the main internal bus. When high, real-time data is selected. | | |
| TP0 | P9 | I | Test pins. High during normal operation. (See Table 6). | | |
| TP1 | R10 | | | | |
| VCC | C7 | | 5-V supply | | |
| VCC | C12 | | | | |
| VCC | D3 | | | | |
| VCC | D8 | | | | |
| VCC | H4 | | | | |
| VCC | H12 | | | | |
| VCC | M8 | | | | |
| VCC | M13 | | | | |
| VCC | N4 | | | | |
| VCC | N9 | | | | |
| WE | A6 | I | Write enable for control flip-flops, active low | | |

overview

The 64 I/O pins of the 'ACT8841A are arranged in 16 nibble groups of four bits each, where each set of four pins serves as bidirectional inputs to and outputs from a nibble multiplexer. During a switching operation, each nibble passes four bits of either stored or real-time data to the main internal 64-bit data bus. Each output multiplexer will independently select one of the 16 nibbles from this 64-bit data bus.

Data nibbles are organized into two groups: the least significant half (D31-D0) and the most significant half (D63-D32). Stored versus real-time data inputs can be selected separately for the LSH and the MSH. Two clock inputs, LSCLK and MSCLK, are available to latch LSH and MSH data inputs, respectively, into the data register.

The pattern of output nibbles resulting from the switching operation is determined by a selectable control source, either one of eight banks of programmable control flip-flops or one of two hard-wired switching configurations. Inputs to the control flip-flops can be loaded either from the data bus or from control I/Os. A separate clock (CRCLK) is provided for loading the banks of control flip-flops.

architecture

The 'ACT8841A digital crossbar switch has its 64 data I/Os arranged in 16 multiplexer logic blocks, as shown in the functional block diagram. Each nibble multiplexer logic block handles four bits of real-time input and four bits of stored-data input, and either input can be passed to the common data bus.

Two input multiplexer controls are provided to select between stored and real-time inputs. SELDLS controls input data selection for the LSH (D31-D0) of the 64-bit data input, and SELDMS for the MSH (D63-D32). The input register clocks, LSCLK and MSCLK, are grouped in the same way and are used to clock data into the registers in the multiplexer logic blocks. The 16 data input nibbles make up the 64 data bits on the internal main bus.

This common bus supplies 16 data nibbles to a 16-to-1 output multiplexer in each multiplexer logic block (see Figure 3). As determined by one of ten selectable control sources, the 16-to-1 output multiplexer selects a data nibble to send to the outputs via the three-state output driver.

Control of the input and output multiplexers determines the input-to-output pattern for the entire crossbar switch. Many different switching combinations can be set up by programming the control flip-flop configurations to determine the outputs from the 16-to-1 multiplexers.

For example, the switch can be programmed to broadcast one data input nibble through the other 15 nibbles (60 outputs). Conversely, a 15-to-1 nibble multiplexer can be configured by programming the switch to select and output a single data nibble from the 64-bit bus. Several examples are described in more detail in a later section.

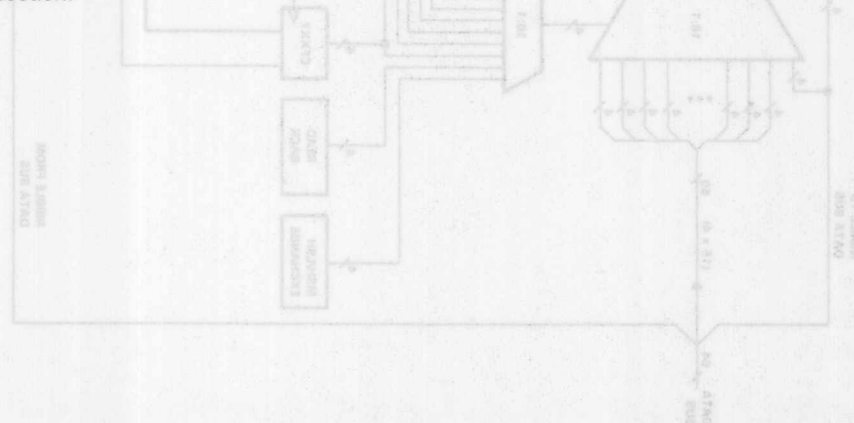


FIGURE 1. DATA NIBBLE MULTIPLEXER LOGIC

multiplexer logic group

There are 16 multiplexer logic blocks, one for each nibble. External data flows from four data I/O pins into a logic block. A block diagram of the multiplexer logic is shown in Figure 1. The data inputs are either clocked into the data register or passed directly to the main internal bus. The 64 bits of data from the main bus are presented to a 16-to-1 multiplexer, which selects the data nibble output.

Each of the 16 nibble multiplexer logic blocks contains eight control flip-flop (CF) groups, one for each of the control banks. A control bank stores one complete switching configuration. Each CF group consists of four D-type edge-triggered flip-flops. In Figure 1, the CF groups are shown as CFXX0 to CFXX7, where XX indicates the number of the nibble multiplexer logic group ($0 \leq XX \leq 15$). CFXX0 represents the 16 CF groups (one from each logic block) that make up flip-flop control bank 0, CFXX1 the 16 CF groups in bank 1, etc.

In addition to the eight banks of programmable flip-flops, two hard-wired switching configurations can be selected. The MSH/LSH exchange directs the input nibbles from each half of the switch to the data outputs directly opposite. This switching pattern is shown in Table 1 below. For example, data input on D11-D8 is output on D43-D40, and data input on D43-D40 is output on D11-D8.

Table 1. MSH/LSH Exchange

| LSH | | MSH |
|---------|---|---------|
| D3-D0 | ↔ | D35-D32 |
| D7-D4 | ↔ | D39-D36 |
| D11-D8 | ↔ | D43-D40 |
| D15-D12 | ↔ | D47-D44 |
| D19-D16 | ↔ | D51-D48 |
| D23-D20 | ↔ | D55-D52 |
| D27-D24 | ↔ | D59-D56 |
| D31-D28 | ↔ | D63-D60 |

The second hard-wired configuration, a read-back function, causes all 64 bits to be output on the same I/Os on which they were input. Neither of the hard-wired control configurations affects the contents of the control banks.

The control source select, CRSEL3-CRSEL0, determines which switching pattern is selected, as shown in Table 2.

Table 2. 16-to-1 Output Multiplexer Control Source Selects

| CRSEL3 | CRSEL2 | CRSEL1 | CRSEL0 | CONTROL SOURCE SELECTED |
|--------|--------|--------|--------|----------------------------------|
| L | L | L | L | Control bank 0 (programmable) |
| L | L | L | H | Control bank 1 (programmable) |
| L | L | H | L | Control bank 2 (programmable) |
| L | L | H | H | Control bank 3 (programmable) |
| L | H | L | L | Control bank 4 (programmable) |
| L | H | L | H | Control bank 5 (programmable) |
| L | H | H | L | Control bank 6 (programmable) |
| L | H | H | H | Control bank 7 (programmable) |
| H | X | X | L | MSH/LSH exchange* |
| H | X | X | H | Read-back (output echoes input)* |

*Hard-wired switching configuration
X = don't care

SN74ACT8841A
DIGITAL CROSSBAR SWITCH

control words

A CF group can store a four-bit control word (CFN3-CFN0) to select the output of the 16-to-1 multiplexer for that nibble port. One control word is loaded in each CF group. A total of 16 words, one per multiplexer logic block, are loaded in a bank to configure one complete switching pattern. Table 3 lists the control words and the input data each selects.

Each control word can be stored in a CF group and sent as an internal control signal to select the output of a 16-to-1 multiplexer in a nibble logic block. For example, any CF group loaded with the word "LHHH" will select the data input on D31-D28 as the outputs of the associated nibble. If all 16 CF groups in a bank were loaded with "LHHH," the same output (D31-D28) would be selected by the entire switch.

Table 3. 16-to-1 Output Multiplexer Control Words

| INTERNAL SIGNALS | | | | INPUT DATA SELECTED AS |
|------------------|------|------|------|------------------------|
| CFN3 | CFN2 | CFN1 | CFN0 | MULTIPLEXER OUTPUT |
| L | L | L | L | D3-D0 |
| L | L | L | H | D7-D4 |
| L | L | H | L | D11-D8 |
| L | L | H | H | D15-D12 |
| L | H | L | L | D19-D16 |
| L | H | L | H | D23-D20 |
| L | H | H | L | D27-D24 |
| L | H | H | H | D31-D28 |
| H | L | L | L | D35-D32 |
| H | L | L | H | D39-D36 |
| H | L | H | L | D43-D40 |
| H | L | H | H | D47-D44 |
| H | H | L | L | D51-D48 |
| H | H | L | H | D55-D52 |
| H | H | H | L | D59-D56 |
| H | H | H | H | D63-D60 |

loading control configurations

CRWRITE2-CRWRITE0 select which control bank is being loaded, as shown in Table 4.

Table 4. Control Flip-Flops Load Destination Select

| CRWRITE2 | CRWRITE1 | CRWRITE0 | DESTINATION |
|----------|----------|----------|----------------|
| L | L | L | Control bank 0 |
| L | L | H | Control bank 1 |
| L | H | L | Control bank 2 |
| L | H | H | Control bank 3 |
| H | L | L | Control bank 4 |
| H | L | H | Control bank 5 |
| H | H | L | Control bank 6 |
| H | H | H | Control bank 7 |

The control words for a bank can be loaded either 16 bits at a time on the control I/O pins (CNTR15-CNTR0) or all 64 bits at once on the data inputs (D63-D0). If the control load source select, CRSRCE, is high, the words are loaded from the data inputs. When CRSRCE = L, the CNTR inputs are used.

When a control bank is loaded from the data inputs, \overline{WE} , CRSRCE, CRWRITE2-CRWRITE0, and the control register clock CRCLK are used in combination to load all 16 control words (64 bits) in a single cycle. A MSH/LSH exchange like that shown in Table 3 is used to load the flip flops on a rising CRCLK clock edge. For example, data inputs D3-D0 go to the data bus and then to the CF group that selects the data outputs for D35-D32. CRWRITE2-CRWRITE0 select the control bank that is loaded (see Table 6).

The CNTR15-CNTR0 inputs can also be used to load the control banks. The bank is selected by CRWRITE2-CRWRITE0 (see Table 6). Four control words per CRCLK cycle can be input to the CF groups (CFXX) that make up the bank. The CF groups loaded are selected by CRADR1-CRADR0, as shown in Table 5. Four CRCLK cycles are needed to load an entire control bank.

Table 5. Loading Control Flip-Flops from CNTR I/Os

| CRAD1 | CRAD0 | \overline{WE} | CRCLK | CF GROUPS LOADED BY CONTROL (CNTR) I/O NUMBERS | | | |
|-------|-------|-----------------|--------------|---|------|-----|-----|
| | | | | 15-12 | 11-8 | 7-4 | 3-0 |
| L | L | L | \downarrow | CF12 | CF8 | CF4 | CF0 |
| L | H | L | \downarrow | CF13 | CF9 | CF5 | CF1 |
| H | L | L | \downarrow | CF14 | CF10 | CF6 | CF2 |
| H | H | L | \downarrow | CF15 | CF11 | CF7 | CF3 |
| X | X | H | X | Inhibit write to flip-flops | | | |

To read out the control settings, the same address signals can be used, except that no CRCLK signal is needed and \overline{OEC} is pulled low. CREAD2-CREAD0 select the bank to be read; the format is the same as for CRWRITE2-CRWRITE0, shown in Table 4.

Using the control I/Os to read the control bank settings can be valuable during debugging or diagnostics. Control settings are volatile and will be lost if the 'ACT8841A is powered off. An external program controlling switch operation may need to read the control bank settings so that it can save and restore the current switching configurations.

test pins

TP1-TPO test pins are provided for system testing. As Table 6 shows, these pins should be maintained high during normal operation. To force all outputs and I/Os low, low signals are placed on TP1-TPO and all output enables ($\overline{OED15-OED0}$ and \overline{OEC}). To force all outputs and I/Os high, TP1 and all output enables are pulled low, and TPO is driven high. When TPO is left low and a high signal is placed on TP1, all outputs on the 'ACT8841A are placed in a high-impedance state, isolating the chip from the rest of the system.

Table 6. Test Pin Inputs

| TP1 | TPO | $\overline{OED15-OED0}$ | \overline{OEC} | RESULT |
|-----|-----|-------------------------|------------------|--|
| L | L | L | L | All outputs and I/Os forced low |
| L | H | L | L | All outputs and I/Os forced high |
| H | L | X | X | All outputs placed in a high-impedance state |
| H | H | X | X | Normal operation (default state) |

SN74ACT8841A

DIGITAL CROSSBAR SWITCH

examples

Most ACT8841A switch configurations are straightforward to program, involving few control signals and procedures to set up the control words in the banks of flip-flops. Control signals and procedures for loading and using control words are shown in the following examples.

broadcasting a nibble

Any of the 16 data input nibbles can be broadcast to the other 15 data nibbles for output. For ease of presentation, input nibble D63-D60 is used in this example. Example 1 presents the microcode sequence for loading flip-flop bank 0 and executing the nibble broadcast.

The low signal on CRSRCE selects CNTR15-CNTR0 as the input source, and the low signals on CRWRITE2-CRWRITE0 select flip-flop bank 0 as the destination. Table 3 shows that to select data on D63-D60 as the output nibble, the four bits in the control word CFN3-CFN0 must be high; therefore the CNTR15-CNTR0 inputs are coded high. The four microcode instructions shown in Example 1 load the same control word from CNTR15-CNTR0 into all 16 CF groups of bank 0.

Once the control flip-flops have been loaded, the switch can be used to broadcast nibble D63-D60 as programmed. The microcode instruction to execute the broadcast is shown as the last instruction in Example 1. WE is held high and the data to be broadcast is input on D63-D60. The high signal on SELDMS selects a real-time data input for the broadcast. MSCLK and LSCLK (not shown) can be used to load the input registers if the input nibble is to be retained. No register clock signals are needed if the input data is not being stored.

The banks of control flip-flops not selected as a control source can be loaded with new control words or read out on CNTR15-CNTR0 while the switch is operating. For example, the MSH data inputs can be used to load flip-flop bank 1 of the LSH while bank 0 of the LSH is controlling data I/O.

To read out the control settings, the same address signals can be used, except that no CRCLK signal is needed and DEC is pulled low. CREAD3-CREAD0 select the bank to be read; the format is the same as for CRWRITE2-CRWRITE0, shown in Table 4.

Using the control I/Os to read the control bank settings can be valuable during debugging or diagnostics. Control settings are volatile and will be lost if the ACT8841A is powered off. An external program controlling switch operation may need to read the control bank settings so that it can save and restore the current switching configurations.

TR1-TPO test pins are provided for system testing. As Table 6 shows, these pins should be maintained high during normal operation. To force all outputs and I/Os low, low signals are placed on TR1-TPO and I/O output enables (OED15-OED0 and DEC). To force all outputs and I/Os high, TR1 and all output enables are pulled low, and TPO is driven high. When TPO is left low and a high signal is placed on TR1, all outputs are placed in a high-impedance state, isolating the chip from the rest of the system.

Table 6. Test Pin Inputs

| TR1 | TPO | OED15-OED0 | DEC | RESULT |
|-----|-----|------------|-----|--|
| L | L | L | L | All outputs and I/Os forced low |
| L | H | L | L | All outputs and I/Os forced high |
| H | L | X | X | All outputs placed in a high-impedance state |
| H | H | X | X | Normal operation (default state) |

Example 1. Programming a Nibble Broadcast

| INST. NO. | CRSRCE | CRWRITE2 | CRWRITE1 | CRWRITE0 | CRADR1 | CRADR0 | CNTR I/O NUMBERS | | | | CRSEL3 | CRSEL2 | CRSEL1 | CRSEL0 | \overline{WE} | SELDMS | SELDLS | OED15-OED0 | | | | OEC | CRCLK |
|-----------|--------|----------|----------|----------|--------|--------|------------------|------|------|------|--------|--------|--------|--------|-----------------|--------|--------|------------|------|------|------|-----|-------|
| | | | | | | | 15-12 | 11-8 | 7-4 | 3-0 | | | | | | | | | | | | | |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1111 | 1111 | 1111 | 1111 | X | X | X | X | 0 | X | X | XXXX | XXXX | XXXX | XXXX | 1 | |
| 2 | 0 | 0 | 0 | 0 | 0 | 1 | 1111 | 1111 | 1111 | 1111 | X | X | X | X | 0 | X | X | XXXX | XXXX | XXXX | XXXX | 1 | |
| 3 | 0 | 0 | 0 | 0 | 1 | 0 | 1111 | 1111 | 1111 | 1111 | X | X | X | X | 0 | X | X | XXXX | XXXX | XXXX | XXXX | 1 | |
| 4 | 0 | 0 | 0 | 0 | 1 | 1 | 1111 | 1111 | 1111 | 1111 | X | X | X | X | 0 | X | X | XXXX | XXXX | XXXX | XXXX | 1 | |
| 5 | X | X | X | X | X | X | XXXX | XXXX | XXXX | XXXX | 0 | 0 | 0 | 0 | 1 | 1 | X | 1000 | 0000 | 0000 | 0000 | 1 | None |

Comments

| INST. NO. | COMMENT |
|-----------|---|
| 1 | Loads CF12, CF8, CF4, CF0 of bank 0 |
| 2 | Loads CF13, CF9, CF5, CF1 of bank 0 |
| 3 | Loads CF14, CF10, CF6, CF2 of bank 0 |
| 4 | Loads CF15, CF11, CF7, CF3 of bank 0 |
| 5 | Selects bank 0 for switching control Selects real-time data inputs |

Example 2. Programming an MSH/LSH Exchange on CNTR Inputs

| INST. NO. | CRSRCE | CRWRITE2 | CRWRITE1 | CRWRITE0 | CRADR1 | CRADR0 | CNTR I/O NUMBERS | | | | CRSEL3 | CRSEL2 | CRSEL1 | CRSEL0 | \overline{WE} | SELDMS | SELDLS | OED15-OED0 | | | | OEC | CRCLK |
|-----------|--------|----------|----------|----------|--------|--------|------------------|------|------|------|--------|--------|--------|--------|-----------------|--------|--------|------------|------|------|------|-----|-------|
| | | | | | | | 15-12 | 11-8 | 7-4 | 3-0 | | | | | | | | | | | | | |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0100 | 0000 | 1100 | 1000 | X | X | X | X | 0 | X | X | XXXX | XXXX | XXXX | XXXX | 1 | |
| 2 | 0 | 1 | 1 | 1 | 0 | 1 | 0101 | 0001 | 1101 | 1001 | X | X | X | X | 0 | X | X | XXXX | XXXX | XXXX | XXXX | 1 | |
| 3 | 0 | 1 | 1 | 1 | 1 | 0 | 0111 | 0011 | 1111 | 1011 | X | X | X | X | 0 | X | X | XXXX | XXXX | XXXX | XXXX | 1 | |
| 4 | 0 | 1 | 1 | 1 | 1 | 1 | 0111 | 0011 | 1111 | 1011 | X | X | X | X | 0 | X | X | XXXX | XXXX | XXXX | XXXX | 1 | |
| 5 | X | X | X | X | X | X | XXXX | XXXX | XXXX | XXXX | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0000 | 0000 | 0000 | 0000 | 1 | None |

Comments

| INST. NO. | COMMENT |
|-----------|--|
| 1 | Loads CF12, CF8, CF4, CF0 of bank 7 |
| 2 | Loads CF13, CF9, CF5, CF1 of bank 7 |
| 3 | Loads CF14, CF10, CF6, CF2 of bank 7 |
| 4 | Loads CF15, CF11, CF7, CF3 of bank 7 |
| 5 | Selects bank 7 for switching control Selects registered data inputs |

programming an MSH/LSH exchange

A second, more complicated example involves programming the switch to swap corresponding nibbles between the MSH and the LSH (first nibble in the LSH for first nibble in the MSH, and so on). This swap can be implemented using the hard-wired logic circuit selected when CRSEL3 is high and CRSEL0 is low. Programming this swap without using the MSH/LSH exchange logic requires loading a different control word into each mux logic block. This is described below for purposes of illustration.

Each nibble in one half, either LSH or MSH, selects as output the registered data from the corresponding nibble in the other half. The registered data from D35-D32 is to be output on D3-D0, the registered data from D3-D0 is output on D35-D32, and so on for the remaining nibbles. As shown in Table 2, the flip-flops for D3-D0 have to be set to 1000 and the D35-D32 inputs must be low. The CF groups and control words involved in this switching pattern are listed in Table 7.

Table 7. Control Words for an MSH/LSH Exchange

| CF GROUP | CNTR INPUTS TO LOAD FLIP-FLOPS | CONTROL WORD LOADED | RESULTS |
|----------|--------------------------------|---------------------|-------------------|
| CF15 | CNTR15- CNTR12 | 0111 | D31-D28 ↔ D63-D60 |
| CF14 | | 0110 | D27-D24 ↔ D59-D56 |
| CF13 | | 0101 | D23-D20 ↔ D55-D52 |
| CF12 | | 0100 | D19-D16 ↔ D51-D48 |
| CF11 | CNTR11- CNTR8 | 0011 | D15-D12 ↔ D47-D44 |
| CF10 | | 0010 | D11-D8 ↔ D43-D40 |
| CF9 | | 0001 | D7-D4 ↔ D39-D36 |
| CF8 | | 0000 | D3-D0 ↔ D35-D32 |
| CF7 | CNTR7- CNTR4 | 1111 | D63-D60 ↔ D31-D28 |
| CF6 | | 1110 | D59-D56 ↔ D27-D24 |
| CF5 | | 1101 | D55-D52 ↔ D23-D20 |
| CF4 | | 1100 | D51-D48 ↔ D19-D16 |
| CF3 | CNTR3- CNTR0 | 1011 | D47-D44 ↔ D15-D12 |
| CF2 | | 1010 | D43-D40 ↔ D11-D8 |
| CF1 | | 1001 | D39-D36 ↔ D7-D4 |
| CF0 | | 1000 | D35-D32 ↔ D3-D0 |

With this list of control words and the signals in Table 5, the 16-bit control inputs on CNTR15-CNTR0 can be arranged to load the control flip-flops in four cycles. Example 2 shows the microcode instructions for loading the control words and executing the exchange.

In Example 2, bank 7 of flip-flops is being programmed. Bank 7 is selected by taking CRWRITE2-CRWRITE0 high and leaving CRSRCE low (see Table 4) when the control words are loaded on CNTR15-CNTR0. With WE held low, the CRCLK is used to load the four sets of control words. Once the flip-flops are loaded, data can be input on D63-D0 and the programmed pattern of output selection can be executed. A microinstruction to select registered data inputs and bank 7 as the control source is shown as the last instruction in Example 2. The data must be clocked into the input registers, using LSCLK and MSCLK, before the last instruction is executed.

The control flip-flops could also have been loaded from the data input nibbles in one CRCLK cycle. Input nibbles from one half are mapped onto the control flip-flops of the other half. All control words to set up a switching pattern should be loaded before the bank of flip-flops is selected as control source. The microcode instructions to load bank 1 with the 16 control words in one cycle are presented in Example 3.

Example 3. Loading the MSH/LSH Exchange from Data Inputs

| CRSRCE | CRWRITE2 | CRWRITE1 | CRWRITE0 | WE | SELDMS | SELDLS | OED15-OED0 | CRCLK |
|--------|----------|----------|----------|----|--------|--------|---------------------|-------|
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1111 1111 1111 1111 | |

These control nibbles may be loaded from the input as a 64-bit real-time input word or as two 32-bit words stored previously. To use stored control words, MSCLK and LSCLK are used to load the LSH and MSH input registers with the correct sequence of control nibbles. Whenever the flip-flops are loaded from the data inputs, all 64 bits of control data must be present when the CRCLK is used so that all control nibbles in a program are loaded simultaneously. Example 4 presents the three microcode instructions to load the MSH and LSH input registers and then to pass the registered data to flip-flop bank 2.

Example 4. Loading Control Flip-Flops from Input Registers

| INST. NO. | CRSRCE | CRWRITE2 | CRWRITE1 | CRWRITE0 | WE | SELDMS | SELDLS | OED15-OED0 | CRCLK | MSCLK | LSCLK | COMMENTS |
|-----------|--------|----------|----------|----------|----|--------|--------|------------|-------|-------|-------|---------------------|
| 1 | X | X | X | X | 1 | X | X | 1 | None | | None | Load inputs D63-D32 |
| 2 | X | X | X | X | 1 | X | X | 1 | None | None | | Load inputs D31-D0 |
| 3 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | | None | None | Load control bank 2 |

The control words in a program can also be read back from the flip-flops using the CNTR outputs. Four instructions are necessary to read the 64 bits in a bank of flip-flops out on CNTR15-CNTR0. WE is held high and OEC is taken low. No CRCLK signal is required. CREAD2-CREAD0 select bank 2 of flip-flops, and CRADR1-CRADR0 select in sequence the four addresses of the 16-bit words to be read out on the CNTR outputs. Example 5 shows the four microcode instructions.

Example 5. Reading Control Settings on CNTR Outputs

| INST. NO. | CREAD2 | CREAD1 | CREAD0 | OEC | CRADR1 | CRADR0 | WE | CNTR I/O NUMBERS | | | | COMMENT |
|-----------|--------|--------|--------|-----|--------|--------|----|------------------|------|------|------|---------------------------|
| | | | | | | | | 15-12 | 11-8 | 7-4 | 3-0 | |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0100 | 0000 | 1100 | 1000 | Read CF12, CF8, CF4, CF0 |
| 2 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0101 | 0001 | 1101 | 1001 | Read CF13, CF9, CF5, CF1 |
| 3 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0110 | 0010 | 1110 | 1010 | Read CF14, CF10, CF6, CF2 |
| 4 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0111 | 0011 | 1111 | 1011 | Read CF15, CF11, CF7, CF3 |

SN74ACT8841A DIGITAL CROSSBAR SWITCH

absolute maximum ratings over operating free-air temperature range (unless otherwise noted)[†]

| | |
|--|----------------|
| Supply voltage, V_{CC} (see Note 1) | -0.5 V to 6 V |
| Input clamp current, I_{IK} ($V_I < 0$ or $V_I > V_{CC}$) | ± 20 mA |
| Output clamp current, I_{OK} ($V_O < 0$ or $V_O > V_{CC}$) | ± 50 mA |
| Continuous output current, I_O ($V_O = 0$ to V_{CC}) | ± 50 mA |
| Continuous current through V_{CC} or GND pins | ± 100 mA |
| Operating free-air temperature range | 0°C to 70°C |
| Storage temperature range | -65°C to 150°C |

[†]Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTE 1: All voltage levels are with respect to GND.

recommended operating conditions

| PARAMETER | MIN | NOM | MAX | UNIT |
|--|------|-----|----------|------|
| V_{CC} Supply voltage | 4.75 | 5.0 | 5.25 | V |
| V_{IH} High-level input voltage | 2 | | V_{CC} | V |
| V_{IL} Low-level input voltage | 0 | | 0.8 | V |
| I_{OH} High-level output current | | | -8 | mA |
| I_{OL} Low-level output current | | | 8 | mA |
| V_I Input voltage | 0 | | V_{CC} | V |
| V_O Output voltage | 0 | | V_{CC} | V |
| dt/dv Input transition rise or fall rate | 0 | | 15 | ns/V |
| T_A Operating free-air temperature | 0 | | 70 | °C |

electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

| PARAMETER | TEST CONDITIONS | V_{CC} | MIN | TYP [‡] | MAX | UNIT |
|----------------------|--|----------|------|------------------|---------|---------|
| V_{OH} | $I_{OH} = -20 \mu A$ | 4.75 V | 4.6 | | | V |
| | | 5.25 V | 5.1 | | | |
| | $I_{OH} = -8 \text{ mA}$ | 4.75 V | 3.85 | 3.95 | | |
| | | 5.25 V | 4.60 | 4.70 | | |
| V_{OL} | $I_{OL} = 20 \mu A$ | 4.75 V | | | 0.1 | V |
| | | 5.25 V | | | 0.1 | |
| | $I_{OL} = 8 \text{ mA}$ | 4.75 V | | 0.32 | 0.45 | |
| | | 5.25 V | | 0.32 | 0.45 | |
| I_I^{\ddagger} | $V_I = V_{CC}$ or 0 | 5.25 V | | ± 0.1 | ± 1 | μA |
| I_{CCQ} | $V_I = V_{CC}$ or 0 | 5.25 V | | 100 | 200 | μA |
| C_I | $V_I = V_{CC}$ or 0 | 5 V | | 10 | | pF |
| ΔI_{CC}^{\S} | One input at 3.4 V, other inputs at 0 or V_{CC} | 5.25 V | | | 1 | mA |

[‡]For I/O ports, the parameters I_{OZH} and I_{OZL} include the offstate output current.

^{\S}This is the increase in supply current for each input that is at one of the specified TTL voltage levels rather than 0 or V_{CC} .

^{\P}All typical values are at $V_{CC} = 5 \text{ V}$, $T_A = 25^\circ \text{C}$.



POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

SN74ACT8841A

DIGITAL CROSSBAR SWITCH

switching characteristics over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)[†]

| PARAMETER | FROM | TO | MIN | TYP [‡] | MAX | UNIT |
|-----------|------------------|-------------|-----|------------------|-----|------|
| t_{pd} | Data in | Data out | | 7 | 14 | ns |
| | MSCLK, LSCLK | | | 10 | 18 | |
| | SELDMS, SELDLS | | | 9 | 15 | |
| | CRCLK | | | 12 | 19 | |
| | CRSEL3-CRSELO | CNTRn | | 12 | 19 | ns |
| | CREAD2-CREADO | | | 10 | 18 | |
| | CRCLK | | | 10 | 18 | |
| | CRAD1, CRADO | | | 8 | 16 | |
| t_{en} | TP1, TP0 | All outputs | | 10 | 19 | ns |
| | TP1, TP0 | All outputs | | 10 | 15 | |
| | \overline{OED} | Data out | | 7 | 12 | |
| t_{dis} | \overline{OEC} | CNTRn | | 8 | 14 | ns |
| | TP1, TP0 | All outputs | | 10 | 15 | |
| | \overline{OED} | Data out | | 5 | 8 | |
| | \overline{OEC} | CNTRn | | 6 | 10 | |

[†] See Parameter Measurement Information for load circuit and voltage waveforms.

[‡] All typical values are at $V_{CC} = 5\text{ V}$, $T_A = 25^\circ\text{C}$.

timing requirements over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

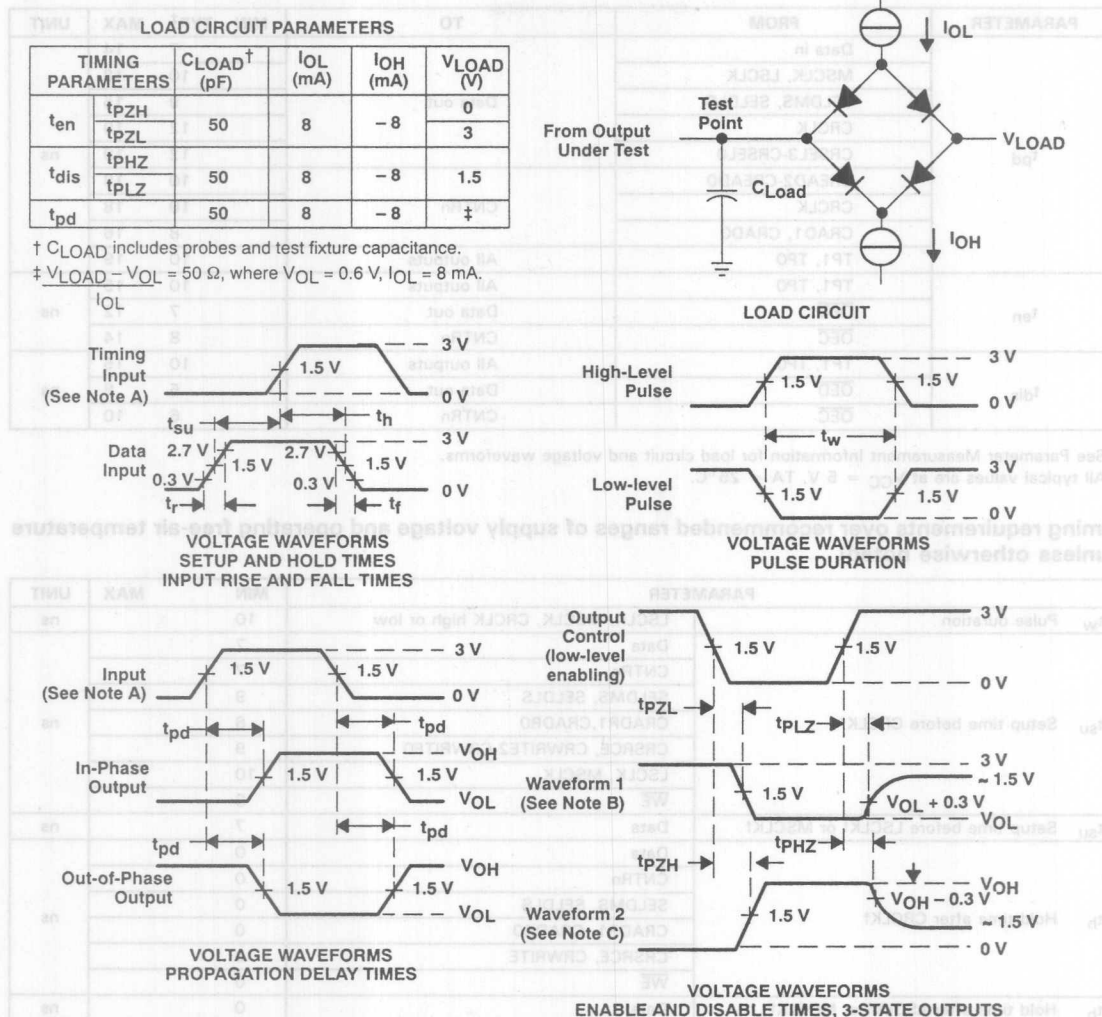
| PARAMETER | MIN | MAX | UNIT |
|---|---------------------------|-----|------|
| t_w Pulse duration | 10 | | ns |
| t_{su} Setup time before CRCLK [†] | Data | 7 | ns |
| | CNTRn | 7 | |
| | SELDMS, SELDLS | 9 | |
| | CRADR1, CRADRO | 8 | |
| | CRSRCE, CRWRITE2-CRWRITE0 | 9 | |
| | LSCLK, MSCLK | 10 | |
| t_{su} Setup time before LSCLK [†] or MSCLK [†] | WE | 8 | ns |
| | Data | 7 | |
| t_h Hold time after CRCLK [†] | Data | 0 | ns |
| | CNTRn | 0 | |
| | SELDMS, SELDLS | 0 | |
| | CRADR1, CRADRO | 0 | |
| | CRSRCE, CRWRITE | 0 | |
| | WE | 0 | |
| t_h Hold time after LSCLK [†] or MSCLK [†] | Data | 0 | ns |



SN74ACT8841A

DIGITAL CROSSBAR SWITCH

PARAMETER MEASUREMENT INFORMATION



- Notes:
- A. Phase relationships between waveforms were chosen arbitrarily. All input pulses are supplied by pulse generators having the following characteristics: $PRR = 1 MHz$, $Z_0 = 50 \Omega$, $t_r \leq 6 ns$, $t_f \leq 6 ns$.
 - B. Waveform 1 is for an output with internal conditions such that the output is low except when disabled by the output control.
 - C. Waveform 2 is for an output with internal conditions such that the output is high except when disabled by the output control. For t_{PLZ} and t_{PHZ} , V_{OL} and V_{OH} are specified values.

FIGURE 2

TYPICAL APPLICATION INFORMATION

'AS8840 AND 'ACT8841A FUNCTIONAL COMPARISON

The 'ACT8841A and the bipolar SN74AS8840 share the same architecture. Microcode for the 'AS8840 can be run on the 'ACT8841A if the additional control inputs to the 'ACT8841A are properly terminated. However, because the 'ACT8841A is a CMOS device with six additional control inputs, the 'AS8840 and the 'ACT8841A are not socket-compatible and cannot be used interchangeably.

differences between the SN74AS8840 and the SN74ACT8841A

The SN74AS8840 and the SN74ACT8841A digital crossbar switches essentially perform the same function. The 'AS8840 and the 'ACT8841A are based on the same 16-port architecture, differing in the number of control registers, power consumption, and pin-out.

One difference is in the number of programmable control flip-flop banks available to configure the switch. The 'AS8840 has two programmable control banks, while the 'ACT8841A has eight. Both have two selectable hard-wired switching configurations.

The increased number of control banks in the 'ACT8841A require six additional pins not found on the 'AS8840. These are: CRWRITE2, CRWRITE1, CREAD2, CREAD1, CRSEL3, and CRSEL2. CREAD and CRWRITE on the 'AS8840 become CREAD0 and CRWRITE0 on the 'ACT8841A. On the 'AS8840, CRSEL1 selects the hardwired control functions when high. This function is performed by the CRSEL3 signal on the 'ACT8841A. Therefore, CRSEL2 and CRSEL1 are actually the added signals.

The 'ACT8841A is a low-power CMOS device requiring only 5-V power. Because of its STL internal logic and TTL I/Os, the 'AS8840 requires both 2-V and 5-V power.

Both the 'AS8840 and the 'ACT8841A are in 156-pin grid-array packages; however, the two devices are not pin-for-pin compatible. Control signals were added to the 'ACT8841A and the 2-V V_{CC} pins of the 'AS8840 were assigned other functions in the 'ACT8841A.

changing 'AS8840 microcode to 'ACT8841A microcode

Since only six signals have been added to the 'ACT8841A, changing existing 'AS8840 microcode to 'ACT8841A microcode is straight forward. CRSEL3 on the 'ACT8841A is functionally equivalent to CRSEL1 on the 'AS8840. CREAD2, CREAD1, CRWRITE2, CRWRITE1, CRSEL2, and CRSEL1 bits must be added. These can always be 0 if no additional control banks are needed. Additional control configurations can be stored by programming these bits.

All other signals in the 'AS8840 microcode remain the same when converting to 'ACT8841A microcode.

TYPICAL APPLICATION INFORMATION

'A58840 AND 'ACT8841A FUNCTIONAL COMPARISON

The 'ACT8841A and the bipolar SN74A58840 share the same architecture. Microcode for the 'A58840 can be run on the 'ACT8841A if the additional control inputs to the 'ACT8841A are properly terminated. However, because the 'ACT8841A is a CMOS device with six additional control inputs, the 'A58840 and the 'ACT8841A are not socket-compatible and cannot be used interchangeably.

Differences between the SN74A58840 and the SN74ACT8841A

The SN74A58840 and the SN74ACT8841A digital crossbar switches essentially perform the same function. The 'A58840 and the 'ACT8841A are based on the same 16-port architecture, differing in the number of control registers, power consumption, and pin-out.

One difference is in the number of programmable control flip-flop banks available to configure the switch. The 'A58840 has two programmable control banks, while the 'ACT8841A has eight. Both have two selectable hard-wired switching configurations.

The increased number of control banks in the 'ACT8841A reduce six additional pins not found on the 'A58840. These are: CRWRITE2, CRWRITE1, CREAD2, CREAD1, CRSEL3, and CRSEL2. CREAD and CRWRITE on the 'A58840 become CREAD0 and CRWRITE0 on the 'ACT8841A. On the 'A58840, CRSEL1 selects the hardwired control function when high. This function is performed by the CRSEL3 signal on the 'ACT8841A. Therefore, CRSEL2 and CRSEL1 are actually the added signals.

The 'ACT8841A is a low-power CMOS device requiring only 5-V power. Because of its STL internal logic and TTL I/Os, the 'A58840 requires both 3-V and 5-V power.

Both the 'A58840 and the 'ACT8841A are in 156-pin gull-array packages; however, the two devices are not pin-for-pin compatible. Control signals were added to the 'ACT8841A and the 3-V VCC pins of the 'A58840 were assigned other functions in the 'ACT8841A.

changing 'A58840 microcode to 'ACT8841A microcode

Since only six signals have been added to the 'ACT8841A, changing existing 'A58840 microcode to 'ACT8841A microcode is straightforward. CRSEL3 on the 'ACT8841A is functionally equivalent to CRSEL1 on the 'A58840. CREAD0, CREAD1, CRWRITE0, CRWRITE1, CRSEL2, and CRSEL1 bits must be added. These can always be 0 if no additional control banks are needed. Additional control configurations can be stored by programming these bits.

All other signals in the 'A58840 microcode remain the same when converting to 'ACT8841A microcode.

| | |
|---|-----------|
| General Information | 1 |
| SN74ACT8818A 16-Bit Microsequencer | 2 |
| SN74ACT8832A 32-Bit Registered ALU | 3 |
| SN74ACT8836A 32- × 32-Bit Parallel Multiplier | 4 |
| SN74ACT8841A Digital Crossbar Switch | 5 |
| SN74ACT8847 64-Bit Floating-Point/Integer Unit | 6 |
| SN74ACT8847 Application Information | 7 |
| SN74ACT8867 32-Bit Vector Processor Unit | 8 |
| Design Support | 9 |
| Mechanical Data | 10 |

| | |
|----|--|
| 1 | General Information |
| 2 | SN7A4CT8818A 16-Bit Microsequencer |
| 3 | SN7A4CT8832A 32-Bit Registered ALU |
| 4 | SN7A4CT8830A 32 x 32-Bit Parallel Multiplier |
| 5 | SN7A4CT8841A Digital Crossbar Switch |
| 6 | SN7A4CT8847 64-Bit Floating-Point/Integer Unit |
| 7 | SN7A4CT8847 Application Information |
| 8 | SN7A4CT8867 32-Bit Vector Processor Unit |
| 9 | Design Support |
| 10 | Mechanical Data |

SN74ACT8847 64-BIT FLOATING-POINT UNIT

D3377, DECEMBER 1989—REVISED APRIL 1990

- Meets IEEE Standard for Single- and Double-Precision Formats
- Performs Floating-Point and Integer Add, Subtract, Multiply, Divide, Square Root, and Compare
- Multiplier and ALU May Operate in Parallel
- Performs Logical Operations and Logical Shifts
- 64-Bit IEEE Divide in 11 Cycles, 64-Bit Square Root in 14 Cycles
- 30-ns SN74ACT8847-30 and 40-ns SN74ACT8847-40 Pipelined Performance for 66 MFLOPS and 50 MFLOPS, Respectively
- Low-Power 0.8- μ m EPIC™ CMOS Process

description

The SN74ACT8847 is a high-speed, double-precision floating-point and integer processor. It performs high-accuracy, scientific computations as part of a customized host processor or as a powerful stand-alone device. Its advanced math processing capabilities allow the chip to accelerate the performance of both CISC- and RISC-based systems.

High-end computer systems, such as graphics workstations, mini-computers and 32-bit personal computers, can utilize the single-chip 'ACT8847 for both floating-point and integer functions.

The SN74ACT8847 combines a multiplier and an arithmetic-logic unit in a single microprogrammable VLSI device. The 'ACT8847 is implemented in Texas Instruments 0.8- μ m CMOS technology to offer high speed and low power consumption with exceptional flexibility and functional integration. The FPU can be microprogrammed to operate in multiple modes to support a variety of floating-point applications.

The 'ACT8847 is fully compatible with the IEEE Std 754-1985 for binary floating-point arithmetic. This FPU performs both single- and double-precision operations, integer operations, logical operations, and division and square root operations (as single microinstructions).

understanding the 'ACT8847 floating-point unit

To support floating-point processing in IEEE format, the 'ACT8847 may be configured for either single- or double-precision operation. Instruction inputs can be used to select three modes of operation, including independent ALU operations, independent multiplier operations, or simultaneous ALU and multiplier operations.

Three levels of internal data registers are available. The device can be used in pipelined mode (all registers enabled) or in other available register configurations. An instruction register, a 64-bit constant register, and a status register are also provided.

Each FPU can handle three types of data input formats. The ALU accepts data operands in integer format or IEEE floating-point format. A third type of operand, denormalized numbers, can also be processed after the ALU has converted them to "wrapped" numbers, which are explained in detail in a later section. The 'ACT8847 multiplier operates on normalized floating-point numbers, wrapped numbers, and integer operands.

microprogramming the 'ACT8847

The 'ACT8847 is a fully microprogrammable device. Each FPU operation is specified by a microinstruction or sequence of microinstructions which set up the control inputs of the FPU so that the desired operation is performed.

EPIC is a trademark of Texas Instruments Incorporated.

PRODUCTION DATA documents contain information current as of publication date. Products conform to specifications per the terms of Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.


**TEXAS
INSTRUMENTS**
POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

Copyright © 1990, Texas Instruments Incorporated

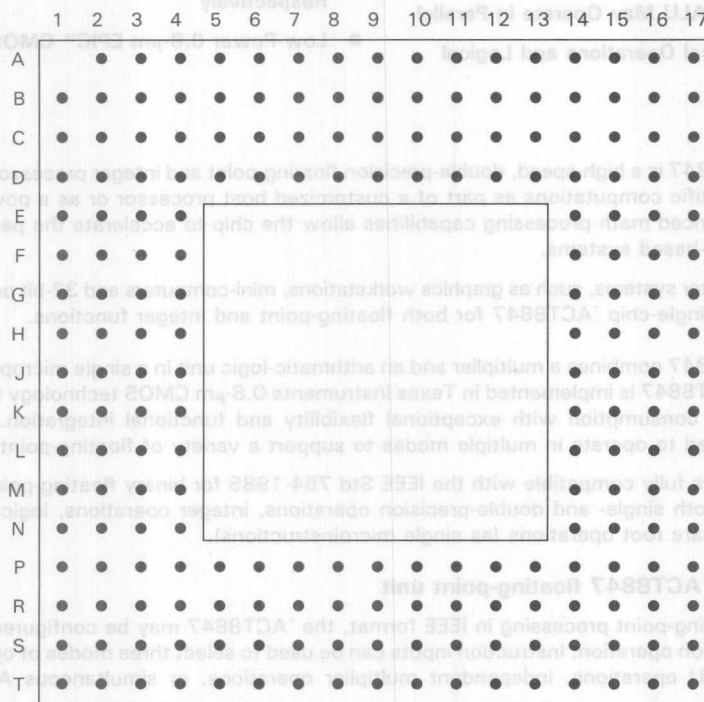
SN74ACT8847 **64-BIT FLOATING-POINT UNIT**

pin descriptions

Pin descriptions and grid assignment for the 'ACT8847 are given on the following pages. The pin at location A1 has been omitted for indexing purposes.

207 PIN . . . GA PACKAGE

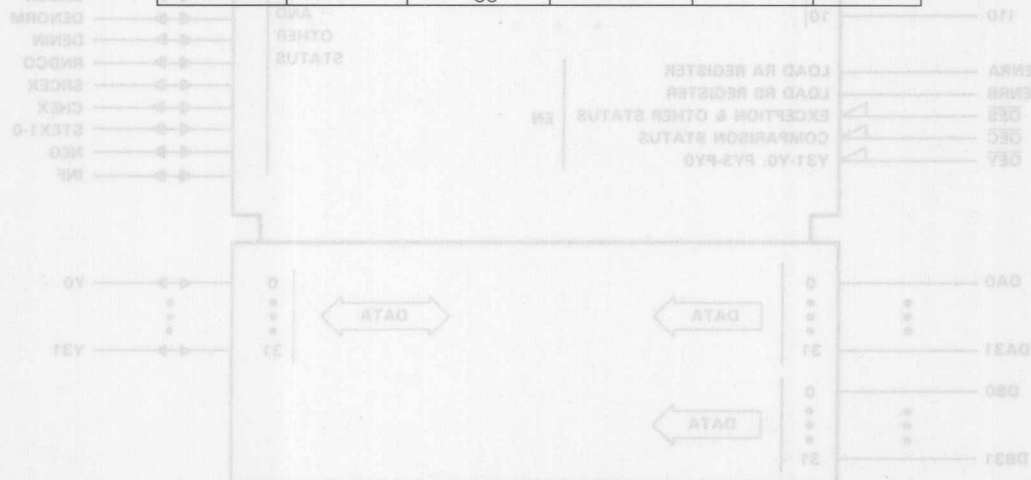
(TOP VIEW)



SN74ACT8847
64-BIT FLOATING-POINT UNIT

PIN GRID ASSIGNMENT

| PIN NO. NAME | PIN NO. NAME | PIN NO. NAME | PIN NO. NAME | PIN NO. NAME | PIN NO. NAME |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| A1 missing | C2 Y0 | E3 FAST | J15 FLOWC | P1 ENRC | S1 NC |
| A2 INF | C3 Y3 | E4 GND | J16 SRCC | P2 PIPES0 | S2 PB0 |
| A3 Y5 | C4 Y6 | E14 GND | J17 BYTEP | P3 RESET | S3 DB0 |
| A4 Y8 | C5 Y9 | E15 AGTB | K1 SELOP3 | P4 PB1 | S4 DB4 |
| A5 Y11 | C6 Y12 | E16 AEQB | K2 SELOP4 | P5 DB1 | S5 DB11 |
| A6 Y14 | C7 Y15 | E17 MSERR | K3 SELOP5 | P6 DB5 | S6 DB12 |
| A7 Y17 | C8 Y18 | F1 I5 | K4 GND | P7 DB9 | S7 DB15 |
| A8 Y20 | C9 Y23 | F2 I3 | K14 GND | P8 DB16 | S8 DB19 |
| A9 Y21 | C10 Y26 | F3 RND0 | K15 PA1 | P9 DB21 | S9 DB23 |
| A10 Y24 | C11 Y30 | F4 GND | K16 PA2 | P10 DB28 | S10 DB26 |
| A11 Y27 | C12 PY1 | F14 GND | K17 PA3 | P11 DA0 | S11 DB30 |
| A12 Y29 | C13 UNDER | F15 PERRA | L1 SELOP6 | P12 DA4 | S12 DA2 |
| A13 PY0 | C14 INEX | F16 OEY | L2 SELOP7 | P13 DA8 | S13 DA6 |
| A14 PY3 | C15 DENIN | F17 OES | L3 CLK | P14 DA12 | S14 DA10 |
| A15 IVAL | C16 SRCEX | G1 I7 | L4 VCC | P15 DA19 | S15 DA14 |
| A16 NEG | C17 CHEX | G2 I6 | L14 GND | P16 DA22 | S16 DA15 |
| A17 NC | D1 I1 | G3 I4 | L15 DA30 | P17 DA23 | S17 DA17 |
| B1 ED | D2 RND1 | G4 VCC | L16 DA31 | R1 PIPES1 | T1 NC |
| B2 Y2 | D3 Y1 | G14 VCC | L17 PA0 | R2 HALT | T2 PB3 |
| B3 Y4 | D4 GND | G15 OEC | M1 ENRB | R3 PB2 | T3 DB3 |
| B4 Y7 | D5 VCC | G16 SELMS/LS | M2 ENRA | R4 DB2 | T4 DB7 |
| B5 Y10 | D6 GND | G17 TEST1 | M3 CLKC | R5 DB6 | T5 DB8 |
| B6 Y13 | D7 GND | H1 I10 | M4 GND | R6 DB10 | T6 DB13 |
| B7 Y16 | D8 VCC | H2 I9 | M14 VCC | R7 DB14 | T7 DB17 |
| B8 Y19 | D9 GND | H3 I8 | M15 DA27 | R8 DB18 | T8 DB20 |
| B9 Y22 | D10 GND | H4 GND | M16 DA28 | R9 DB22 | T9 DB24 |
| B10 Y25 | D11 VCC | H14 GND | M17 DA29 | R10 DB27 | T10 DB25 |
| B11 Y28 | D12 GND | H15 TEST0 | N1 CONFIG0 | R11 DB31 | T11 DB29 |
| B12 Y31 | D13 GND | H16 SELST1 | N2 CONFIG1 | R12 DA3 | T12 DA1 |
| B13 PY2 | D14 VCC | H17 SELST0 | N3 CLKMODE | R13 DA7 | T13 DA5 |
| B14 OVER | D15 STEX1 | J1 SELOP2 | N4 PIPES2 | R14 DA11 | T14 DA9 |
| B15 RNDCO | D16 STEX0 | J2 SELOP1 | N14 DA18 | R15 DA16 | T15 DA13 |
| B16 DENORM | D17 UNORD | J3 SELOP0 | N15 DA24 | R16 DA20 | T16 NC |
| B17 DIVBY0 | E1 I2 | J4 VCC | N16 DA25 | R17 DA21 | T17 NC |
| C1 PERRB | E2 IO | J14 VCC | N17 DA26 | | |

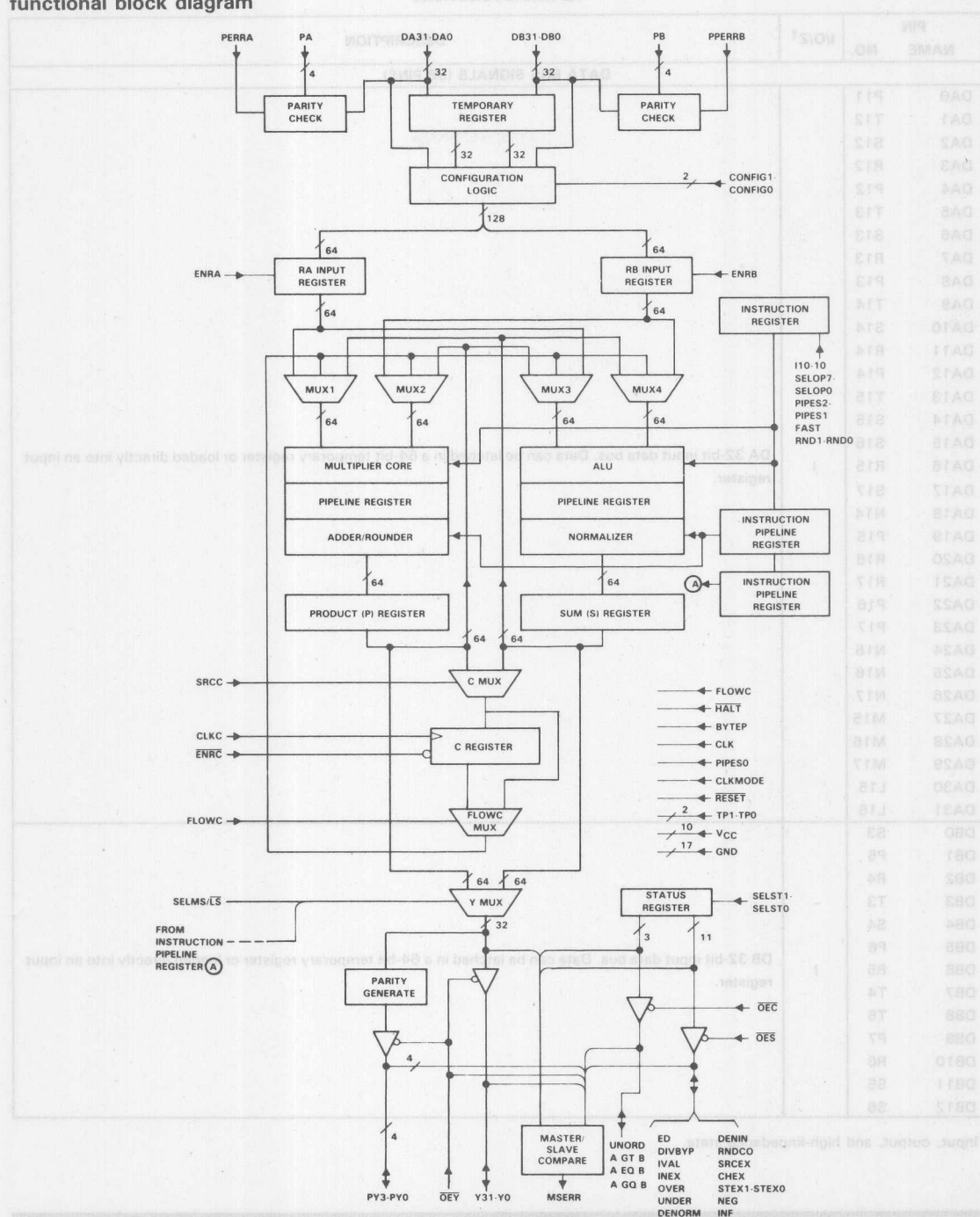


TEXAS
INSTRUMENTS

POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

SN74ACT8847 64-BIT FLOATING-POINT UNIT

functional block diagram



SN74ACT8847
64-BIT FLOATING-POINT UNIT

TERMINAL FUNCTIONS

| PIN NAME | NO. | I/O/Z† | DESCRIPTION |
|----------------------------|-----|--------|---|
| DATA BUS SIGNALS (96 PINS) | | | |
| DA0 | P11 | I | DA 32-bit input data bus. Data can be latched in a 64-bit temporary register or loaded directly into an input register. |
| DA1 | T12 | | |
| DA2 | S12 | | |
| DA3 | R12 | | |
| DA4 | P12 | | |
| DA5 | T13 | | |
| DA6 | S13 | | |
| DA7 | R13 | | |
| DA8 | P13 | | |
| DA9 | T14 | | |
| DA10 | S14 | | |
| DA11 | R14 | | |
| DA12 | P14 | | |
| DA13 | T15 | | |
| DA14 | S15 | | |
| DA15 | S16 | | |
| DA16 | R15 | | |
| DA17 | S17 | | |
| DA18 | N14 | | |
| DA19 | P15 | | |
| DA20 | R16 | | |
| DA21 | R17 | | |
| DA22 | P16 | | |
| DA23 | P17 | | |
| DA24 | N15 | | |
| DA25 | N16 | | |
| DA26 | N17 | | |
| DA27 | M15 | | |
| DA28 | M16 | | |
| DA29 | M17 | | |
| DA30 | L15 | | |
| DA31 | L16 | | |
| DB0 | S3 | I | DB 32-bit input data bus. Data can be latched in a 64-bit temporary register or loaded directly into an input register. |
| DB1 | P5 | | |
| DB2 | R4 | | |
| DB3 | T3 | | |
| DB4 | S4 | | |
| DB5 | P6 | | |
| DB6 | R5 | | |
| DB7 | T4 | | |
| DB8 | T5 | | |
| DB9 | P7 | | |
| DB10 | R6 | | |
| DB11 | S5 | | |
| DB12 | S6 | | |

†Input, output, and high-impedance state.

SN74ACT8847
64-BIT FLOATING-POINT UNIT

TERMINAL FUNCTIONS (Continued)

| PIN NAME | NO. | I/O/Z [†] | DESCRIPTION |
|-----------------------------------|-----|--------------------|---|
| DATA BUS SIGNALS (96 PINS) | | | |
| DB13 | T6 | | |
| DB14 | R7 | | |
| DB15 | S7 | | |
| DB16 | P8 | | |
| DB17 | T7 | | |
| DB18 | R8 | | |
| DB19 | S8 | | |
| DB20 | T8 | | |
| DB21 | P9 | | |
| DB22 | R9 | I | DB 32-bit input data bus. Data can be latched in a 64-bit temporary register or loaded directly into an input register. |
| DB23 | S9 | | |
| DB24 | T9 | | |
| DB25 | T10 | | |
| DB26 | S10 | | |
| DB27 | R10 | | |
| DB28 | P10 | | |
| DB29 | T11 | | |
| DB30 | S11 | | |
| DB31 | R11 | | |
| Y0 | C2 | | |
| Y1 | D3 | | |
| Y2 | B2 | | |
| Y3 | C3 | | |
| Y4 | B3 | | |
| Y5 | A3 | | |
| Y6 | C4 | | |
| Y7 | B4 | | |
| Y8 | A4 | | |
| Y9 | C5 | | |
| Y10 | B5 | | |
| Y11 | A5 | | |
| Y12 | C6 | I/O/Z | 32-bit Y output data bus |
| Y13 | B6 | | |
| Y14 | A6 | | |
| Y15 | C7 | | |
| Y16 | B7 | | |
| Y17 | A7 | | |
| Y18 | C8 | | |
| Y19 | B8 | | |
| Y20 | A8 | | |
| Y21 | C9 | | |
| Y22 | B9 | | |
| Y23 | C9 | | |
| Y24 | A10 | | |
| Y25 | B10 | | |

[†]Input, output, and high-impedance state.



POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

SN74ACT8847 **64-BIT FLOATING-POINT UNIT**

TERMINAL FUNCTIONS (Continued)

| PIN NAME | NO. | I/O/Z† | DESCRIPTION |
|---|-----|--------|---|
| DATA BUS SIGNALS (96 PINS) | | | |
| Y26 | C10 | I/O/Z | 32-bit Y output data bus |
| Y27 | A11 | | |
| Y28 | B11 | | |
| Y29 | A12 | | |
| Y30 | C11 | | |
| Y31 | B12 | | |
| PARITY AND MASTER/SLAVE SIGNALS (16 PINS) | | | |
| BYTEP | J17 | I | When high, selects parity generation for each byte of input (four parity bits for each bus). When low, selects parity generation for whole 32-bit input (one parity bit for each bus). Even parity is used. |
| MSERR | E17 | O | Master/Slave error output pin |
| PA0 | L17 | I | Parity inputs for DA data |
| PA1 | K15 | | |
| PA2 | K16 | | |
| PA3 | K17 | | |
| PB0 | S2 | I | Parity inputs for DB data |
| PB1 | P4 | | |
| PB2 | R3 | | |
| PB3 | T2 | | |
| PERRA | F15 | O | DA data parity error output. When high, signals a byte or word has failed an even parity check. |
| PERRB | C1 | O | DB data parity error output. When high, signals a byte or word has failed an even parity check. |
| PY0 | A13 | I/O/Z | Y port parity data |
| PY1 | C12 | | |
| PY2 | B13 | | |
| PY3 | A14 | | |
| CLOCK, CONTROL, AND INSTRUCTION SIGNALS (46 PINS) | | | |
| CLK | L3 | I | Master clock for all registers except C register |
| CLKC | M3 | I | C register clock |
| CLKMODE | N3 | I | Selects whether temporary register loads only on rising clock edge (CLKMODE = L) or on falling edge (CLKMODE = H). |
| CONFIG0 | N1 | I | Select data sources for RA and RB registers from DA bus, DB bus and temporary register |
| CONFIG1 | N2 | | |
| ENRA | M2 | I | When high, enables loading of RA register on a rising clock edge if the RA register is not disabled (see PIPESO below). |
| ENRB | M1 | I | When high, enables loading of RB register on a rising clock edge if the RB register is not disabled (see PIPESO below). |
| ENRC | P1 | I | When low, enables write to C register when CLKC goes high. |
| FAST | E3 | I | When low, selects gradual underflow (IEEE model). When high, selects sudden underflow, forcing all denormalized inputs and outputs to zero. |
| FLOWC | J15 | I | When high, causes product or sum to bypass C register, so that product or sum appears on the C register output bus. Timing is similar to P register or S register feedback operands. C register remains unchanged. Product or sum may also be simultaneously fed back in usual manner (not through C register). |
| HALT | R2 | I | Stalls operation without altering contents of instruction or data registers (except the CREG, which has a separate write enable). Active low. |
| I0 | E2 | I | Instruction inputs |
| I1 | D1 | | |

† Input, output, and high-impedance state.

TERMINAL FUNCTIONS (Continued)

| PIN NAME | NO. | I/O/Z† | DESCRIPTION |
|--|-----|--------|---|
| CLOCK, CONTROL, AND INSTRUCTION SIGNALS (46 PINS) | | | |
| I2 | E1 | | |
| I3 | F2 | | |
| I4 | G3 | | |
| I5 | F1 | | |
| I6 | G2 | I | Instruction inputs |
| I7 | G1 | | |
| I8 | H3 | | |
| I9 | H2 | | |
| I10 | H1 | | |
| OE \overline{C} | G15 | I | Comparison status output enable. Active low. |
| OE \overline{S} | F17 | I | Exception status and other status output enable. Active low. |
| OE \overline{Y} | F16 | I | Y bus output enable. Active low. |
| PIPES0 | P2 | I | When low, enables instruction register and, depending on setting of ENRA and ENRB, the RA and RB input registers. When high, puts instruction, RA and RB registers in flowthrough mode. |
| PIPES1 | R1 | I | When low, enables pipeline registers in ALU and multiplier. When high, puts pipeline registers in flowthrough mode. |
| PIPES2 | N4 | I | When low, enables status register, product (P) and sum (S) registers. When high, puts status register, P and S registers in flowthrough mode. |
| RESET | P3 | I | Clears internal states, status, and exception disable register. Contents of internal pipeline registers are lost. Does not affect other data registers. Active low. |
| RND0 | F3 | | |
| RND1 | D2 | i | Rounding mode control pins. Select four IEEE rounding modes. |
| RNDC0 | B15 | I/O/Z | When high, indicates the mantissa of a number has been increased in magnitude by rounding. |
| SELOP0 | J3 | | |
| SELOP1 | J2 | | |
| SELOP2 | J1 | | |
| SELOP3 | K1 | I | Select operand sources for multiplier and ALU |
| SELOP4 | K2 | | |
| SELOP5 | K3 | | |
| SELOP6 | L1 | | |
| SELOP7 | L2 | | |
| SELST0 | H17 | | |
| SELST1 | H16 | I | Select status source during chained operation |
| SELMS/ \overline{L} S | G16 | I | When low, selects LSH of 64-bit result to be output on the Y bus. When high, selects MSH of 64-bit result. (No effect on single-precision operations.) |
| SRCC | J16 | I | When low, selects ALU as data source for C register. When high, selects multiplier as data source for C register. |
| TEST0 | H15 | | |
| TEST1 | G17 | I | Test pins. Tied high for normal operation. |
| STATUS SIGNALS (17 PINS) | | | |
| AEQB | E16 | I/O/Z | Comparison status or zero detect pin. When high, indicates that A and B operands are equal during a compare operation in the ALU. If not a compare, a high signal indicates a zero result on the Y bus. |
| AGTB | E15 | I/O/Z | Comparison status pin. When high, indicates that A operand is greater than B operand. |
| CHEX | C17 | I/O/Z | Status pin indicating an exception during a chained function. If I6 is low, indicates the multiplier is the source of an exception. If I6 is high, indicates the ALU is the source of an exception. |

†Input, output, and high-impedance state.

SN74ACT8847 **64-BIT FLOATING-POINT UNIT**

TERMINAL FUNCTIONS (Continued)

| PIN NAME | NO. | I/O/Z† | DESCRIPTION |
|---|-----|--------|--|
| STATUS SIGNALS (17 PINS) | | | |
| DENIN | C15 | I/O/Z | Status pin indicating a denormal input to the multiplier. When DENIN goes high, the STEX pins indicate which port had the denormal input. |
| DENORM | B16 | I/O/Z | Status pin indicating a denormal output from the ALU or a wrapped output from the multiplier. In FAST mode, causes the result to go to zero when DENORM is high. |
| DIVBY0 | B17 | I/O/Z | Status pin indicating an attempted operation involved dividing by zero |
| ED | B1 | I/O/Z | Exception detect status signal representing logical OR of all enabled exceptions in the exception disable register |
| INEX | C14 | I/O/Z | Status pin indicating an inexact output |
| INF | A2 | I/O/Z | Status pin. When high, indicates output value is infinity. |
| IVAL | A15 | I/O/Z | Status pin indicating that an invalid operation or a nonnumber (NaN) has been input to the multiplier or ALU. |
| NEG | A16 | I/O/Z | Status pin. When high, indicates result has negative sign. |
| OVER | B14 | I/O/Z | Status pin indicating that the result is greater the largest allowable value for specified format (exponent overflow). |
| SRCEX | C16 | I/O/Z | Status pin indicating source of exception, either ALU (SRCEX = L) or multiplier (SRCEX = H). |
| STEX0 | D16 | I/O/Z | Status pins indicating that a nonnumber (NaN) or denormal number has been input on A port (STEX1) or B port (STEX0). |
| STEX1 | D16 | I/O/Z | Status pins indicating that a nonnumber (NaN) or denormal number has been input on A port (STEX1) or B port (STEX0). |
| UNDER | C13 | I/O/Z | Status pin indicating that a result is inexact and less than minimum allowable value for format (exponent underflow). |
| UNORD | D17 | I/O/Z | Comparison status pin indicating that the two inputs are unordered because at least one of them is a nonnumber (NaN). |
| SUPPLY AND N/C SIGNALS (33 PINS) | | | |
| VCC | D5 | | 5-V supply voltage pins |
| VCC | D8 | | |
| VCC | D11 | | |
| VCC | D14 | | |
| VCC | G4 | I | |
| VCC | G14 | | |
| VCC | J4 | | |
| VCC | J14 | | |
| VCC | L4 | | |
| VCC | M14 | | |
| GND | D4 | | Ground pins. NOTE: All ground pins should be used and connected. |
| GND | D6 | | |
| GND | D7 | | |
| GND | D9 | | |
| GND | D10 | | |
| GND | D12 | | |
| GND | D13 | | |
| GND | E4 | I | |
| GND | E14 | | |
| GND | F4 | | |
| GND | F14 | | Input, output, and high-impedance state. |
| GND | H4 | | |
| GND | H14 | | |
| GND | K4 | | |
| GND | K14 | | |
| GND | L14 | | |
| GND | M4 | | |

†Input, output, and high-impedance state.

TERMINAL FUNCTIONS (Continued)

| PIN NAME | NO. | I/O/Z† | DESCRIPTION |
|----------------------------------|-----|--------|---|
| SUPPLY AND N/C SIGNALS (33 PINS) | | | |
| NC | A17 | | No internal connection. Pins should be left floating. |
| NC | S1 | | |
| NC | T1 | | |
| NC | T16 | | |
| NC | T17 | | |

†Input, output, and high-impedance state.

PRINCIPLES OF OPERATION

introduction

Designing with the 'ACT8847 floating-point unit (FPU) requires a thorough understanding of computer architectures, microprogramming, and IEEE floating-point arithmetic, as well as a detailed knowledge of the 'ACT8847 itself. This introduction presents a brief overview of the 'ACT8847 and discusses a number of issues when designing and programming with this FPU.

major architectural features

The overall architecture for a floating-point system is determined by a combination of design factors. The principal consideration is the set of performance targets that the floating-point processor has to achieve, usually expressed in terms of clock cycle period, operating mode (vector or scalar), and operand precision (32 bit, 64 bit, or other). Of almost equal importance are design constraints of cost, complexity, chip count, power consumption, and requirements for interfacing to other processors.

The architecture of the 'ACT8847 is optimized to satisfy several processing and interface requirements. The FPU has two 32-bit input buses, the DA and DB data buses, and one 32-bit output bus, the Y bus. This 3-port design provides much greater I/O bus bandwidth than can be achieved by a single-port device (one 32-bit I/O bus). Two single-precision inputs can be simultaneously loaded on the input buses while a result is being output on the Y bus.

Internally, the 'ACT8847 FPU consists of two main functional blocks: the multiplier and the ALU (see Figure 1). Either the multiplier or the ALU can operate independently, or the two functional units can be used simultaneously in "chained" mode. When operating independently, each block of the FPU performs a separate set of arithmetic or logical functions. The multiplier supports multiplication, division, and square roots. The ALU supports addition, subtraction, format conversions, logical operations, and shifts. Integer division and integer square root require both the multiplier and the ALU; the final result comes from the ALU.

In chained mode, a multiplier operation executes in parallel with an ALU operation. Possible examples include calculations of a sum of products (multiply and accumulate) or a product of sums (add and then multiply). The sum of products computation requires a total of four operands: two new inputs to be multiplied, the sum of previous products, and the current product to be added to the sum, as shown in Table 1.

TABLE 1. SUM OF PRODUCTS CALCULATION

| MULTIPLIER OPERATION | ALU OPERATION |
|----------------------|-------------------------------|
| $A \times B$ | — |
| $C \times D$ | $(A \times B) + 0$ |
| $E \times F$ | $(C \times D) + (A \times B)$ |
| . | . |
| . | . |
| . | . |



SN74ACT8847
64-BIT FLOATING-POINT UNIT

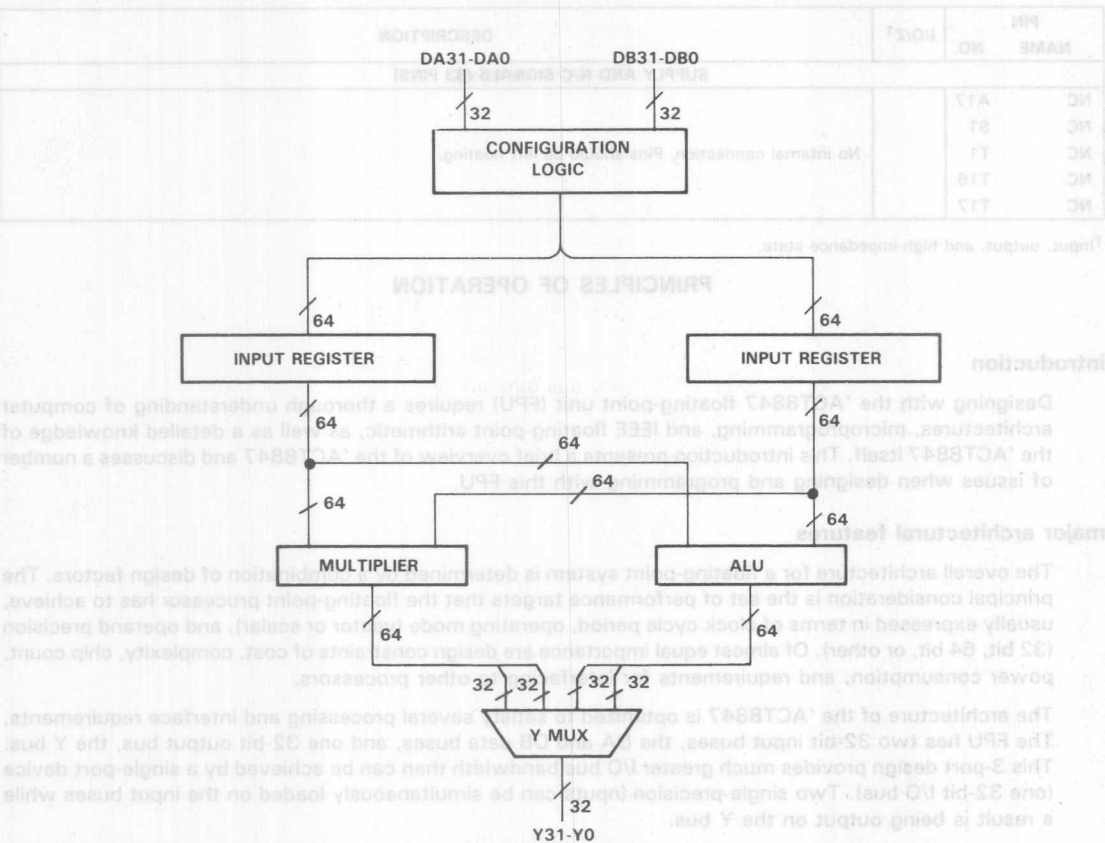


FIGURE 1. HIGH-LEVEL BLOCK DIAGRAM

TABLE 1. SUM OF PRODUCTS CALCULATION

| MULTIPLIER OPERATION | ALU OPERATION |
|----------------------|-------------------------------|
| $A \times B$ | — |
| $C \times D$ | $(A \times B) + C$ |
| $E \times F$ | $(C \times D) + (A \times B)$ |
| ... | ... |
| ... | ... |

Because the 'ACT8847 has multiple internal data paths and data registers, this sum of products can be generated by simultaneous operations on new bus data and internal feedback, without the necessity of storing either the previous accumulation or the current product off chip. Data flow for the sum of products calculation is shown in Figure 2.

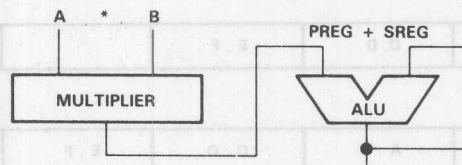


FIGURE 2. MULTIPLY/ACCUMULATE OPERATION

Because the 'ACT8847 has multiple internal data paths and data registers, this sum of products can be generated by simultaneous operations on new bus data and internal feedback, without the necessity of storing either the previous accumulation or the current product off chip. Data flow for the sum of products calculation is shown in Figure 2.

data flow in pipelined architectures

Several levels of internal data registers are available to segment the internal data paths of the 'ACT8847. Enabling one or more registers divides the data paths so that data can be clocked into internal registers, instead of from an external source to an external destination. Enabling the input registers permits data and instruction inputs to be registered on chip. Also, the hardware division and square root operations which the 'ACT8847 performs require that the input registers be enabled.

In the main data paths, three sets of internal registers are available in the 'ACT8847: input registers, pipeline registers in the multiplier and ALU logic blocks, and output registers to capture results from the multiplier and the ALU. When all three levels of data registers are enabled, the register-to-register delay inside the device is minimized. This is the fastest operating mode, and in this configuration the 'ACT8847 is said to be "fully pipelined." While one instruction is executing, the next instruction along with its associated operands may be input to device so that overlapped operations occur (see Figure 3).

The selection of operating mode determines the latency from input to output, the number of clock cycles required for inputs to be processed and results to appear. For each register level enabled in the data path, one clock cycle is added to the latency from input to output.

control architectures for high-speed microprogrammed architectures

A separate control circuit is required to sequence the operation of the 'ACT8847. A sequencer function within the control circuit controls both the sequencer and FPU as determined by FPU status outputs. Either a standard microsequencer such as the SN74ACT8818A, or a custom controller such as a PLA or gate array can be used to control the FPU. Figure 4 shows an example block diagram for a PLA control circuit.

If a standard microsequencer is used, execution addresses for routines stored in the microprogram memory are generated by the microsequencer. As its name implies, microprogram memory stores the sequences of microinstructions which control FPU execution. The 'ACT8847 can be programmed by generating all control bits in a given microinstruction to select a FPU operation.

SN74ACT8847 64-BIT FLOATING-POINT UNIT

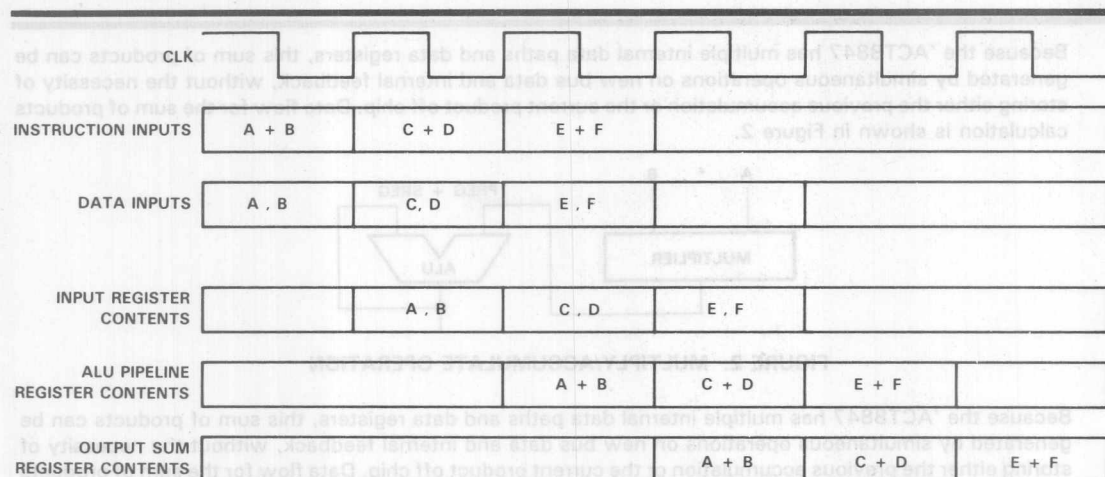


FIGURE 3. EXAMPLE OF FULLY PIPELINED OPERATION

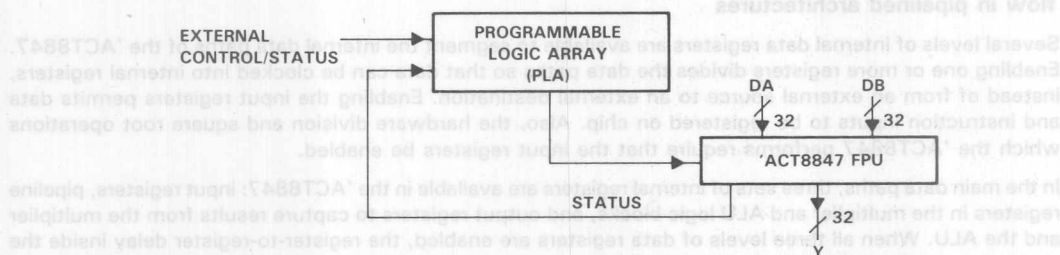


FIGURE 4. PLA CONTROL CIRCUIT EXAMPLE

One possible control circuit for the 'ACT8847 consists of a microsequencer, microprogram memory, and one or more microinstruction registers, together with status logic as required to support a specific floating-point implementation. A control circuit without an instruction register is typically too slow for use with the 'ACT8847. At least one microinstruction register is used to hold the current instruction being executed by the FPU and sequencer (see Figure 5).

Inclusion of the microinstruction register divides the critical path from the sequencer through the program memory to the FPU control inputs, permitting much faster execution times. However, when all the internal registers of the FPU are enabled, FPU operation may be fast enough to require a second register in the control circuit. In this case, a register on the output bus of the sequencer captures each microprogram address, and the microinstruction register captures each microinstruction (see Figure 6).

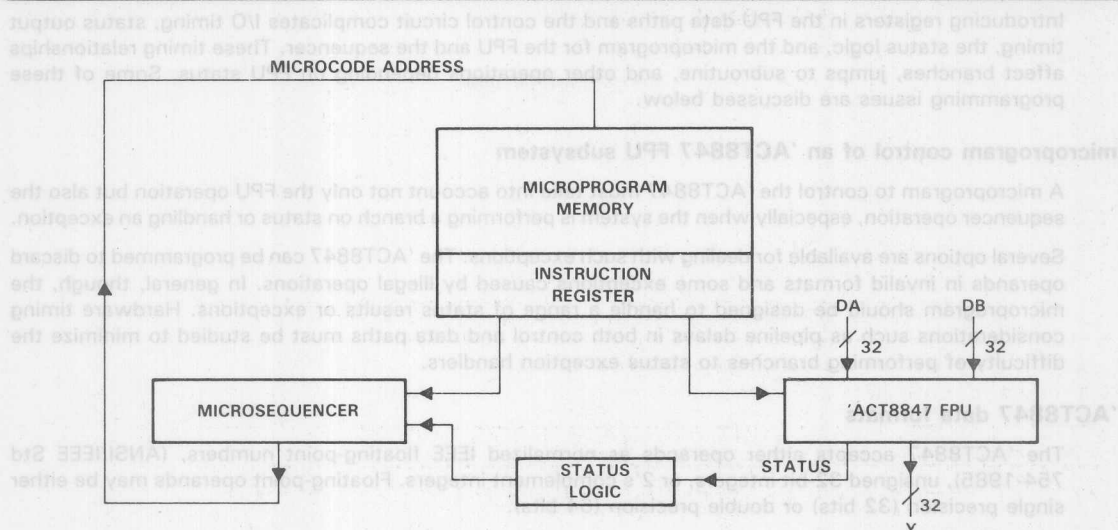


FIGURE 5. MICROPROGRAMMED ARCHITECTURE

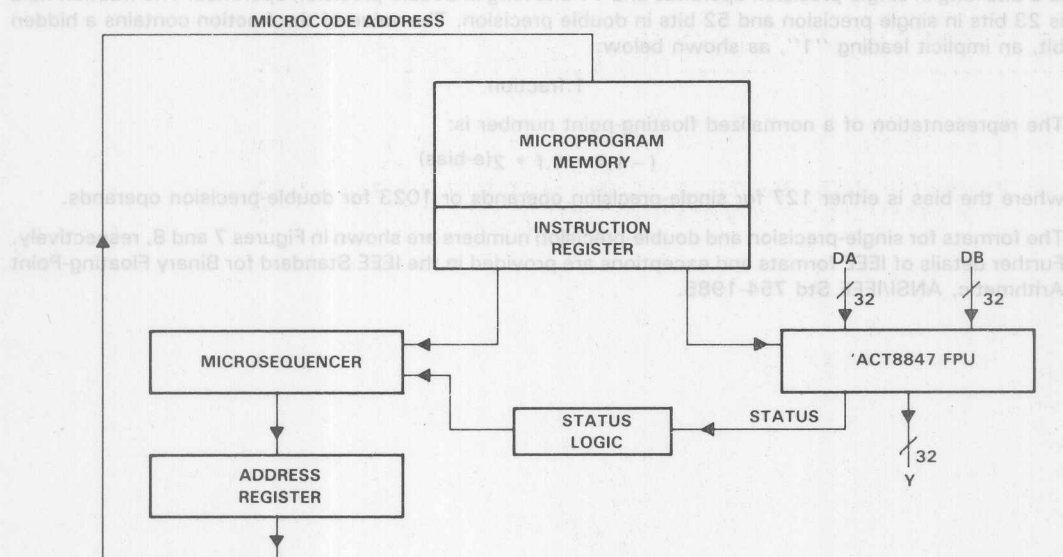


FIGURE 6. MICROPROGRAMMED ARCHITECTURE WITH ADDRESS REGISTER

SN74ACT8847 64-BIT FLOATING-POINT UNIT

Introducing registers in the FPU data paths and the control circuit complicates I/O timing, status output timing, the status logic, and the microprogram for the FPU and the sequencer. These timing relationships affect branches, jumps to subroutine, and other operations depending on FPU status. Some of these programming issues are discussed below.

microprogram control of an 'ACT8847 FPU subsystem

A microprogram to control the 'ACT8847 must take into account not only the FPU operation but also the sequencer operation, especially when the system is performing a branch on status or handling an exception.

Several options are available for dealing with such exceptions. The 'ACT8847 can be programmed to discard operands in invalid formats and some exceptions caused by illegal operations. In general, though, the microprogram should be designed to handle a range of status results or exceptions. Hardware timing considerations such as pipeline delays in both control and data paths must be studied to minimize the difficulty of performing branches to status exception handlers.

'ACT8847 data formats

The 'ACT8847 accepts either operands as normalized IEEE floating-point numbers, (ANSI/IEEE Std 754-1985), unsigned 32-bit integers, or 2's complement integers. Floating-point operands may be either single precision (32 bits) or double precision (64 bits).

IEEE formats for floating-point operands, both single and double precision, consist of three fields: sign, exponent, and fraction, in that order. The leftmost (most significant) bit is the sign bit. The exponent field is 8 bits long in single-precision operands and 11 bits long in double-precision operands. The fraction field is 23 bits in single precision and 52 bits in double precision. The value of the fraction contains a hidden bit, an implicit leading "1", as shown below:

1.fraction

The representation of a normalized floating-point number is:

$$(-1)^s * 1.f * 2^{(e-bias)}$$

where the bias is either 127 for single-precision operands or 1023 for double-precision operands.

The formats for single-precision and double-precision numbers are shown in Figures 7 and 8, respectively. Further details of IEEE formats and exceptions are provided in the IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1985.

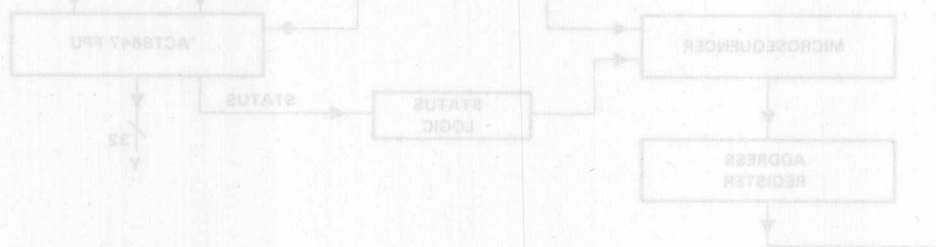


FIGURE 8. MICROPROGRAMMED ARCHITECTURE WITH ADDRESS REGISTER

TABLE 2. IEEE FLOATING-POINT REPRESENTATIONS

| TYPE OF OPERAND | EXPONENT (e) SP (HEX) DP (HEX) | FRACTION (f) HIDDEN | 31 30 23 22 | 0 |
|------------------------------|-----------------------------------|------------------------|-------------|---|
| Normalized Number (max) | FE | | s e f | |
| Normalized Number (min) | 01 | | s e f | |
| Denormalized Number (max) | 00 | | s e f | |
| Denormalized Number (min) | 00 | | s e f | |
| Wrapped Number (max) | 00 | | s e f | |
| Wrapped Number (min) | EA | | s e f | |
| Zero | 00 | | s e f | |
| Infinity | FF | | s e f | |
| NaN (Not a Number) | FF | | s e f | |

FIGURE 7. IEEE SINGLE-PRECISION FORMAT

| 63 62 52 51 | 0 |
|-------------|---|
| s e f | |

FIGURE 8. IEEE DOUBLE-PRECISION FORMAT

The 'ACT8847 also handles two other operand formats which permit operations with very small floating-point numbers. The ALU accepts denormalized floating-point numbers, that is, floating-point numbers so small that they could not be normalized. If these denormal operands are input to the multiplier, they will cause status exceptions. Denormals can be passed through the ALU to be "wrapped" and the wrapped operands can then be input to the multiplier.

A denormalized input has the form of a floating-point number with a zero exponent, a nonzero mantissa, and a zero in the leftmost bit of the mantissa (hidden or implicit bit). Using single precision, a denorm is equal to:

$$(-1)^s * (2)^{-126} * \text{fraction}$$

For double precision, a denorm is equal to:

$$(-1)^s * (2)^{-1022} * \text{fraction}$$

A denormalized number results from decrementing the biased exponent field to zero before normalization is complete. Since a denormalized number cannot be input to the multiplier, it must first be converted to a wrapped number by the ALU. A wrapped number is a number created by normalizing a denormalized number's fraction field and subtracting from the exponent the number of shift positions (minus one) required to do so. The exponent is encoded as a two's complement negative number. When the mantissa of the denormal is normalized by shifting it left, the exponent field decrements from all zeros (wraps past zero) to a negative two's complement number (except in the case of 0.1XXX..., where the exponent is not decremented).

Floating-point formats handled by the 'ACT8847 are presented in Table 2.

SN74ACT8847 **64-BIT FLOATING-POINT UNIT**

TABLE 2. IEEE FLOATING-POINT REPRESENTATIONS

| TYPE OF OPERAND | EXPONENT (e) | | FRACTION (f) (BINARY) | HIDDEN BIT | VALUE OF NUMBER REPRESENTED | |
|------------------------------|--------------|----------|--------------------------|---------------|--------------------------------------|------------------------------------|
| | SP (HEX) | DP (HEX) | | | SP (DECIMAL) [†] | DP (DECIMAL) [†] |
| Normalized Number (max) | FE | 7FE | All 1's | 1 | $(-1)^s (2^{127}) (2 - 2^{-23})$ | $(-1)^s (2^{1023}) (2 - 2^{-52})$ |
| Normalized Number (min) | 01 | 001 | All 0's | 1 | $(-1)^s (2^{-126}) (1)$ | $(-1)^s (2^{-1022}) (1)$ |
| Denormalized Number (max) | 00 | 000 | All 1's | 0 | $(1 - 1)^s (2^{-126}) (1 - 2^{-23})$ | $(-1)^s (2^{-1022}) (1 - 2^{-52})$ |
| Denormalized Number (min) | 00 | 000 | 000...001 | 0 | $(-1)^s (2^{-126}) (2^{-23})$ | $(-1)^s (2^{-1022}) (2^{-52})$ |
| Wrapped Number (max) | 00 | 000 | All 1's | 1 | $(-1)^s (2^{-127}) (2 - 2^{-23})$ | $(-1)^s (2^{-1023}) (2 - 2^{-52})$ |
| Wrapped Number (min) | EA | 7CD | All 0's | 1 | $(-1)^s (2 - (22 + 127)) (1)$ | $(-1)^s (2 - (51 + 1023)) (1)$ |
| Zero | 00 | 000 | Zero | 0 | $(-1)^s (0.0)$ | $(-1)^s (0.0)$ |
| Infinity | FF | 7FF | Zero | 1 | $(-1)^s (\text{infinity})$ | $(-1)^s (\text{infinity})$ |
| NaN (Not a Number) | FF | 7FF | Nonzero | N/A | None | None |

[†]s = sign bit.

status outputs

Status flags are provided to signal both floating-point and integer results. Integer status is provided using AEQB for zero, NEG for sign, and OVER for overflow/carryout.

Status exceptions can result from one or more error conditions such as overflow, underflow, operands in illegal formats, invalid operations, or rounding. Exceptions may be grouped into two classes: input exceptions resulting from invalid operations or denormal inputs to the multiplier, and output exceptions resulting from illegal formats, rounding errors, or both.

architecture overview

The 'ACT8847 is a high-speed floating-point unit implemented in TI's advanced 0.8- μm CMOS technology. The device is fully compatible with IEEE Standard 754-1985 for addition, subtraction, multiplication, division, square root, and comparison.

The 'ACT8847 FPU also performs integer arithmetic, logical operations, and logical shifts. Absolute value conversions, floating-point to integer conversions, and integer to floating-point conversions are also available. The ALU and multiplier are both included in the same device and can be operated in parallel to perform sums of products and products of sums (see the functional block diagram).

IEEE formatted denormal numbers are directly handled by the ALU. Denormal numbers must be wrapped by the ALU before being used in multiplication, division, or square root operations. A fast mode in which all denormals are forced to zero is provided for applications not requiring gradual underflow.

The 'ACT8847 input buses can be configured to operate as two 32-bit data buses or as a single 64-bit bus, providing a number of system interface options. Registers are provided at the inputs, outputs, and inside the ALU and multiplier to support multilevel pipelining. These registers can be bypassed for nonpipelined operation.

A clock mode control allows the temporary input register to be clocked on the rising edge or the falling edge of the clock to support double-precision ALU operations at the same rate as single-precision operations. A feedback register (C register) with a separate clock is provided for temporary internal storage of a multiplier result, ALU result or constant.

Four multiplexers select the multiplier and ALU operands from the input registers, C register, or previous multiplier, or ALU result. Results are output on the 32-bit Y bus; a Y output multiplexer selects the most significant or least significant half of the result if a double-precision number is being output.

To ensure data integrity, parity checking is performed on input data, and parity is generated for output data. A master/slave comparator supports fault-tolerant system design. Two test pin control inputs allow all I/Os and outputs to be forced high, low, or placed in a high-impedance state to facilitate system testing.

pipeline controls

Six data registers in the 'ACT8847 are arranged in three levels along the data paths through the multiplier and the ALU. Each level of registers can be enabled or disabled independently of the other two levels by setting the appropriate PIPES2-PIPES0 inputs. When enabled, data is latched into the register on the rising edge of the system clock (CLK). A separate instruction pipeline register stores the instruction bits corresponding to the operation being executed at each stage.

The levels of pipelining are shown in Figure 9. The first set of registers, the RA and RB input registers, are controlled by PIPES0. These registers may be used as inputs to the ALU, multiplier, or both.

The pipeline registers are the second register set. When enabled by PIPES1, these registers latch intermediate values in the multiplier or ALU.

The results of the ALU and multiplier operations may optionally be latched into two output registers by setting PIPES2 low. The P (product) register holds the result of the multiplier operation; the S (sum) register holds the ALU result.

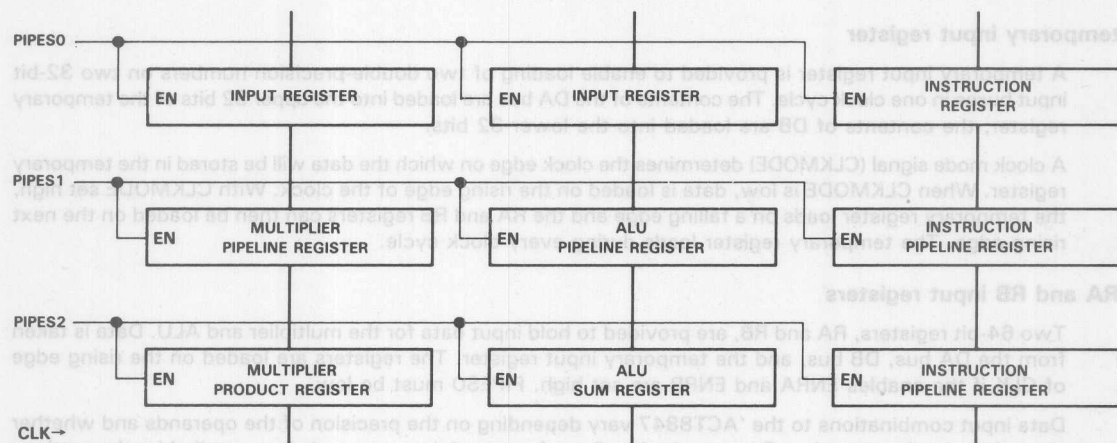


FIGURE 9. PIPELINE CONTROLS

SN74ACT8847

64-BIT FLOATING-POINT UNIT

Table 3 shows the settings of the registers controlled by PIPES2-PIPES0. The instruction pipeline registers are set accordingly.

TABLE 3. PIPELINE CONTROLS (PIPES2-PIPES0)

| PIPES2-PIPES0 | REGISTER OPERATION SELECTED |
|---------------|---|
| X X 0 | Enables input registers (RA, RB) |
| X X 1 | Makes input registers (RA, RB) transparent |
| X 0 X | Enables pipeline registers |
| X 1 X | Makes pipeline registers transparent |
| 0 X X | Enables output registers (PREG, SREG, Status) |
| 1 X X | Makes output registers (PREG, SREG, Status) transparent |

When all registers (except the C register) are enabled, timing constraints can become critical for many double-precision operations. In clock mode 1, the ALU can perform a double-precision operation and output a result during every clock cycle, and both halves of the result must be read out before the end of the next cycle. Status outputs are valid only for the period during which the Y output data is valid.

Similarly, double-precision multiplication is affected by pipelining, clock mode, and sequence of operations. A double-precision multiply may require two cycles to execute and two cycles to output the result, depending on the settings of PIPES2-PIPES0.

Duration of valid outputs at the Y multiplexer depends on settings of PIPES2-PIPES0 and CLKMODE, as well as whether all operations and operands are of the same type. For example, when a double-precision multiply is followed by a single-precision operation, one clock cycle must intervene between the dissimilar operations. The instruction inputs are ignored during this clock cycle.

temporary input register

A temporary input register is provided to enable loading of two double-precision numbers on two 32-bit input buses in one clock cycle. The contents of the DA bus are loaded into the upper 32 bits of the temporary register; the contents of DB are loaded into the lower 32 bits.

A clock mode signal (CLKMODE) determines the clock edge on which the data will be stored in the temporary register. When CLKMODE is low, data is loaded on the rising edge of the clock. With CLKMODE set high, the temporary register loads on a falling edge and the RA and RB registers can then be loaded on the next rising edge. The temporary register loads during every clock cycle.

RA and RB input registers

Two 64-bit registers, RA and RB, are provided to hold input data for the multiplier and ALU. Data is taken from the DA bus, DB bus, and the temporary input register. The registers are loaded on the rising edge of CLK if the enables ENRA and ENRB are set high. PIPES0 must be low.

Data input combinations to the 'ACT8847 vary depending on the precision of the operands and whether they are being input as A or B operands. Loading of external data operands is controlled by the settings of CLKMODE and CONFIG1-CONFIG0, which determine the clock timing for loading and the registers that are used. (See Figure 10).

configuration controls

Three input registers are provided to handle input of data operands, either single precision or double precision. The RA, RB, and temporary registers are each 64 bits wide. The temporary register is (ordinarily) used only during input of double-precision operands.

Double-precision operands are loaded by using the temporary register to store half of the operands prior to inputting the other half of the operands on the DA and DB buses. As shown in Table 4, four configuration modes for selecting input sources are available for loading data operands into the RA and RB registers.

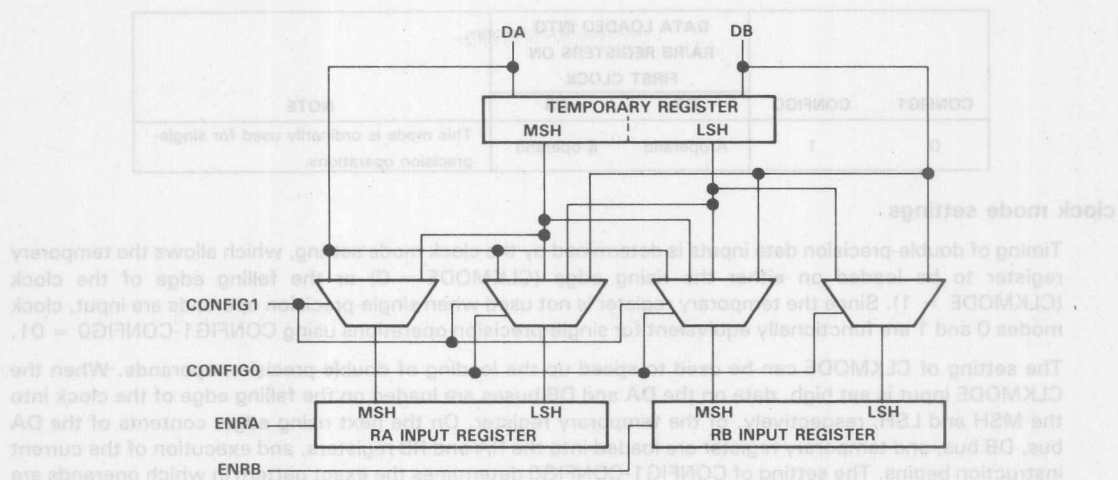


FIGURE 10. INPUT REGISTER CONTROL

TABLE 4. DOUBLE-PRECISION INPUT DATA CONFIGURATION MODES

| | | LOADING SEQUENCE | | | |
|---------|---------|--|-----------------|--|-----------------|
| | | DATA LOADED INTO TEMP REGISTER ON FIRST CLOCK AND RA/RB REGISTERS ON SECOND CLOCK [†] | | DATA LOADED INTO RA/RB REGISTERS ON SECOND CLOCK | |
| CONFIG1 | CONFIG0 | DA | DB | DA | DB |
| 0 | 0 | B operand (MSH) | B operand (LSH) | A operand (MSH) | A operand (LSH) |
| 0 | 1 | A operand (LSH) | B operand (LSH) | A operand (MSH) | B operand (MSH) |
| 1 | 0 | A operand (MSH) | B operand (MSH) | A operand (LSH) | B operand (LSH) |
| 1 | 1 | A operand (MSH) | A operand (LSH) | B operand (MSH) | B operand (LSH) |

[†]On the first active clock edge (see Clock Mode Settings), data in this column is loaded into the temporary register. On the next rising edge, operands in the temporary register and the DA/DB buses are loaded into the RA and RB registers.

When single-precision or integer operands are loaded, the ordinary setting of CONFIG1-CONFIG0 is 01, as shown in Table 5. This setting loads each 32-bit operand in the most significant half (MSH) of its respective register. Single-precision operands are loaded into the MSHs and adjusted to double precision because the data paths internal to the device are all double precision. It is also possible to load single-precision operands with other CONFIG settings but two clock edges are required to load both the A and B operands on the DA bus. The operands are input as the MSHs of the A and B operands (see Table 4). For example, to load single-precision operands using CONFIG1-CONFIG0 = 10, the A and B operands are input one active clock edge before the instruction.

SN74ACT8847
64-BIT FLOATING-POINT UNIT

TABLE 5. SINGLE-PRECISION INPUT DATA CONFIGURATION MODE

| CONFIG1 | CONFIG0 | DATA LOADED INTO RA/RB REGISTERS ON FIRST CLOCK | | NOTE |
|---------|---------|---|-----------|---|
| | | DA | DB | |
| 0 | 1 | A operand | B operand | This mode is ordinarily used for single-precision operations. |

clock mode settings

Timing of double-precision data inputs is determined by the clock mode setting, which allows the temporary register to be loaded on either the rising edge (CLKMODE = 0) or the falling edge of the clock (CLKMODE = 1). Since the temporary register is not used when single-precision operands are input, clock modes 0 and 1 are functionally equivalent for single-precision operations using CONFIG1-CONFIG0 = 01.

The setting of CLKMODE can be used to speed up the loading of double-precision operands. When the CLKMODE input is set high, data on the DA and DB buses are loaded on the falling edge of the clock into the MSH and LSH, respectively, of the temporary register. On the next rising edge, contents of the DA bus, DB bus, and temporary register are loaded into the RA and RB registers, and execution of the current instruction begins. The setting of CONFIG1-CONFIG0 determines the exact pattern in which operands are loaded, whether as MSH or LSH in RA or RB.

Double-precision operation in clock mode 0 is similar except that the temporary register loads only on a rising edge. For this reason, the RA and RB registers do not load until the next rising edge, when all operands are available and execution can begin.

A considerable advantage in speed can be realized by performing double-precision operations with CLKMODE set high. In this clock mode, both double-precision operands can be loaded on successive clock edges, one falling and one rising. If the instruction is an ALU operation, then the operation can be executed in the time from one rising edge of the clock to the next rising edge. Both halves of a double-precision ALU result must be read out on the Y bus within one clock cycle when the ACT8847 is operated in clock mode 1.

The discussion above assumes that the system is able to furnish two sets of operands in one cycle (one set on the falling edge of the clock and the other set on the next rising edge). This assumption may not be valid, since the system is required to "double pump" the input data buses.

Even for a system that is not able to double pump the input data buses, using clock mode 1 can reduce microcode size substantially resulting in increased system throughput. To illustrate, take the case of an operation where the operand(s) are furnished by one or more of the feedback registers (refer to Table 6). Since the input data buses are not being used to furnish the operands, the data on the buses at the time of the instruction is unimportant. By setting CLKMODE high, the instruction begins after the first cycle, resulting in a savings of one cycle.

When single-precision or integer operands are loaded, the original setting of CONFIG1-CONFIG0 is 01, as shown in Table 5. This setting loads each 32-bit operand in the most significant half (MSH) of its respective register. Single-precision operands are loaded into the MSHs and adjusted to double precision because the data paths internal to the device are all double precision. It is also possible to load single-precision operands with other CONFIG settings but two clock edges are required to load both the A and B operands on the DA bus. The operands are input as the MSHs of the A and B operands (see Table 4). For example, to load single-precision operands using CONFIG1-CONFIG0 = 10, the A and B operands are input one active clock edge before the instruction.

TABLE 6(a) DOUBLE-PRECISION CREG + PREG
USING CLKMODE = 0, PIPES2-0 = 010

| CYCLE | CLKMODE | DA BUS | DB BUS | TEMP REG | INSTR BUS | RA REG | RB REG | S REG |
|-------|---------|-----------|-----------|-------------|--------------|-----------|-----------|----------|
| 1 | 0 | X | X | X | C + P | X | X | X |
| 2 | 0 | X | X | X | C + P | X | X | X |
| 3 | X | X | X | X | X | X | X | C + P |

TABLE 6(b) DOUBLE-PRECISION CREG + PREG
USING CLKMODE = 0, PIPES2-0 = 010

| CYCLE | CLKMODE | DA BUS | DB BUS | TEMP REG | INSTR BUS | RA REG | RB REG | S REG |
|-------|---------|-----------|-----------|-------------|--------------|-----------|-----------|----------|
| 1 | 1 | X | X | X | C + P | X | X | X |
| 2 | X | X | X | X | X | X | X | C + P |

Going one step further, take the case of an operation where only one operand needs to be furnished by the input data buses (refer to Table 7). To take advantage of clock mode 1, set the CONFIG lines so that the external operand comes directly from the DA and DB bus, as opposed to coming from the temporary register. Since the temporary register is not used to provide an operand, the data latched into it is inconsequential. It naturally follows then that the clock edge used to load the temporary register is unimportant. So by setting CLKMODE high, a double-precision instruction will begin after one cycle, instead of two cycles.

TABLE 7(a) DOUBLE-PRECISION PREG + RB
USING CLKMODE = 0, PIPES2-0 = 010

| CYCLE | CLKMODE | DA BUS | DB BUS | TEMP REG | INSTR BUS | RA REG | RB REG | S REG |
|-------|---------|-----------|-----------|-------------|--------------|-----------|-----------|----------|
| 1 | 0 | X | X | X | P + RB | X | X | X |
| 2 | 0 | RB(M) | RB(L) | RB | P + RB | X | RB | X |
| 3 | X | X | X | X | X | X | X | P + RB |

TABLE 7(b) DOUBLE-PRECISION PREG + RB
USING CLKMODE = 1, PIPES2-0 = 010

| CYCLE | CLKMODE | DA BUS | DB BUS | TEMP REG | INSTR BUS | RA REG | RB REG | S REG |
|-------|---------|-----------|-----------|-------------|--------------|-----------|-----------|----------|
| 1 | 1 | RB(M) | RB(L) | RB | P + RB | X | RB | X |
| 2 | X | X | X | X | X | X | X | P + RB |

operand selection

Four multiplexers select the multiplier and ALU operands from the RA and RB registers, the previous multiplier or ALU result, or the C register (see Figure 11). The multiplexers are controlled by input signals SELOP7-SELOP0 as shown in Tables 8 and 9. For division and square root operations, operands must be sourced from the input registers RA and RB.

SN74ACT8847 **64-BIT FLOATING-POINT UNIT**

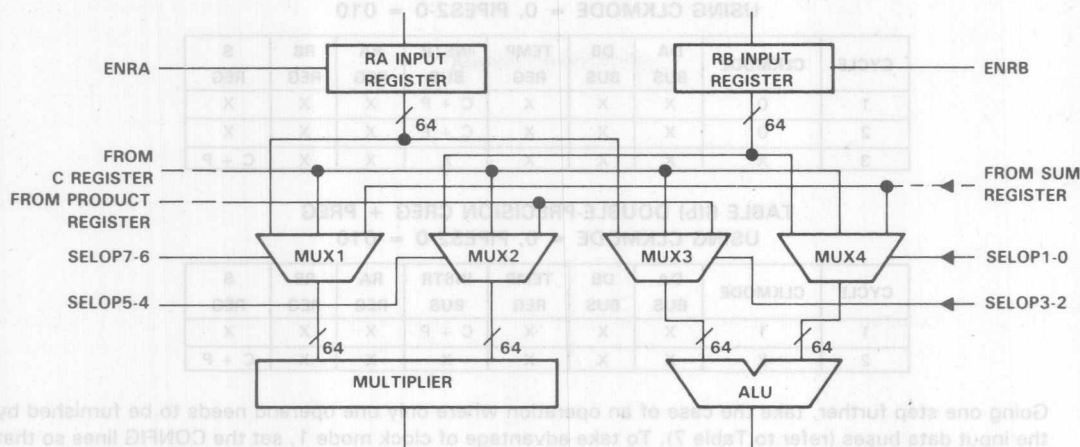


FIGURE 11. OPERAND SELECTION MULTIPLEXER

TABLE 8. MULTIPLIER INPUT SELECTION

| A1 (MUX1) INPUT | | | B1 (MUX2) INPUT | | |
|-----------------|--------|-------------------|-----------------|--------|---------------------|
| SELOP7 | SELOP6 | OPERAND SOURCE† | SELOP5 | SELOP4 | OPERAND SOURCE† |
| 0 | 0 | Reserved | 0 | 0 | Reserved |
| 0 | 1 | C register | 0 | 1 | C register |
| 1 | 0 | ALU feedback | 1 | 0 | Multiplier feedback |
| 1 | 1 | RA input register | 1 | 1 | RB input register |

†For division or square-root operations, only RA and RB registers can be selected as sources.

TABLE 9. ALU INPUT SELECTION

| A1 (MUX1) INPUT | | | B1 (MUX2) INPUT | | |
|-----------------|--------|---------------------|-----------------|--------|-------------------|
| SELOP7 | SELOP6 | OPERAND SOURCE† | SELOP5 | SELOP4 | OPERAND SOURCE† |
| 0 | 0 | Reserved | 0 | 0 | Reserved |
| 0 | 1 | C register | 0 | 1 | C register |
| 1 | 0 | Multiplier feedback | 1 | 0 | ALU feedback |
| 1 | 1 | RA input register | 1 | 1 | RB input register |

† For division or square-root operations, only RA and RB registers can be selected as sources.

As shown in Tables 8 and 9, data operands can be selected from five possible sources, including external inputs from the RA and RB registers, feedback from the P (Product) and S (Sum) registers, and a stored value in the C register. Contents of the C register may be selected as either the A or the B operand in the ALU, the multiplier, or both. When an external input is selected, the RA input always becomes the A operand, and the RB input is the B operand.

Feedback from the ALU can be selected as the A operand to the multiplier or as the B operand to the ALU. Similarly, multiplier feedback may be used as the A operand to the ALU or the B operand to the multiplier. During division or square root operations, operands may not be selected except from the RA and RB input registers (SELOP7-SELOP0 = 11111111).

Selection of operands also interacts with the selected operation in the ALU or the multiplier. ALU operations with one operand are performed only on the A operand (with the exception of the Pass B operation). Also, depending on the instruction selected, the B operand may optionally be forced to zero in the ALU or to one in the multiplier.

If an operation uses one or more feedback registers as operands, the unused bus(es) can be used to preload operand(s) for a later operation. The data is loaded into the RA or RB input register(s); when the data is needed as an operand, the SELOPS pins are set to select the RA or RB register(s), but the register input enables (ENRA, ENRB) are not enabled. The one restriction on preloading data is that the operation being performed during the preload MUST use the same data type (single-precision, double-precision, or integer) as the data being loaded. Operands cannot be preloaded within square root or divide instructions.

C register

The 64-bit constant (C) register is available for storing the result of an ALU or multiplier operation before feedback to the multiplier or ALU. The C register has a separate clock input (CLKC), input source select (SRCC), and write enable (ENRC, active low).

The C register loads from the P or the S register output, depending on the setting of SRCC. $SRCC = 1$ selects the multiplier as the input source. Otherwise, the ALU is selected when $SRCC = 0$. The SRCC input is not registered with the instruction inputs. Depending on the operation selected and the settings of PIPES2-PIPES0, an offset of one or more cycles may be necessary to load the desired result into the C register. The register only loads on a rising edge of CLCK when ENRC is low. (See Figure 12).

A separate control (FLOWC) is available to bypass the C register when feeding an operand back on the C register feedback bus. When FLOWC is high, the output of the P or S register (as selected by SRCC) bypasses the C register without affecting the C register's contents. Direct P or S feedback is unaffected by the FLOWC setting.

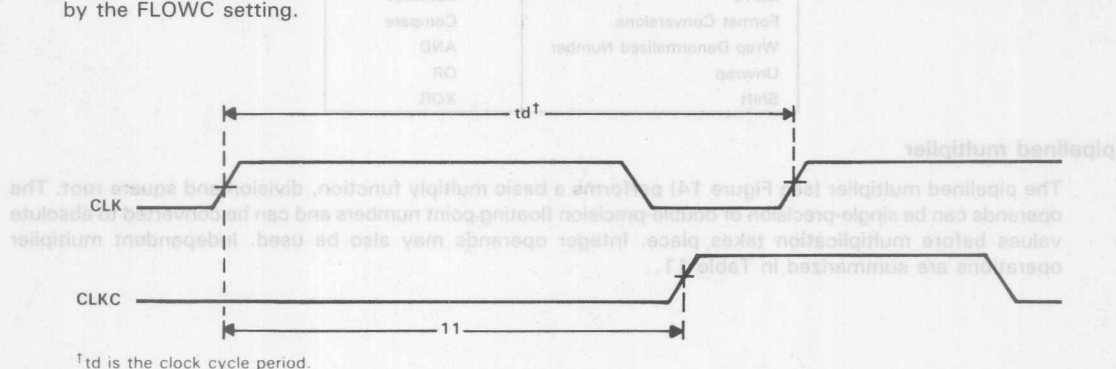


FIGURE 12. C REGISTER TIMING

pipelined ALU

The pipelined ALU contains a circuit for floating-point addition and/or subtraction of aligned operands, a pipeline register, an exponent adjuster and a normalizer/rounder as shown in Figure 13. An exception circuit is provided to detect denormal inputs; these can be flushed to zero if the FAST input is set high. If the FAST input is low, the ALU accepts a denormal as input. A denorm exception flag (DENORM) goes high when the ALU output is a denormal.

Integer processing in the ALU includes both arithmetic and logical operations on either two's complement numbers or unsigned integers. The ALU performs addition, subtraction, comparison, logical shifts, logical AND, logical OR, and logical XOR.

SN74ACT8847 64-BIT FLOATING-POINT UNIT

The ALU may be operated independently or in parallel with the multiplier. Possible ALU functions during independent operation are given in Table 10.

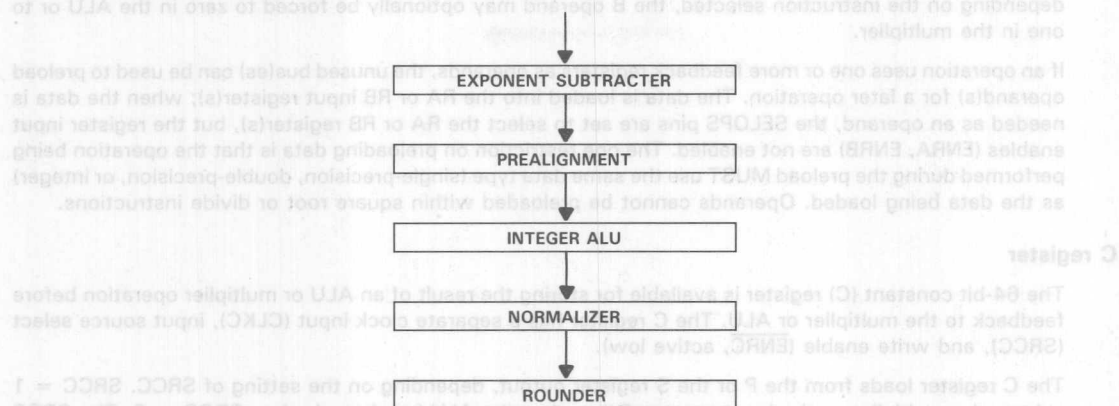


FIGURE 13. FUNCTIONAL DIAGRAM FOR ALU

TABLE 10. INDEPENDENT ALU OPERATIONS

| SINGLE OPERAND | TWO OPERANDS |
|--------------------------|--------------|
| Pass | Add |
| Move | Subtract |
| Format Conversions | Compare |
| Wrap Denormalized Number | AND |
| Unwrap | OR |
| Shift | XOR |

pipelined multiplier

The pipelined multiplier (see Figure 14) performs a basic multiply function, division and square root. The operands can be single-precision or double-precision floating-point numbers and can be converted to absolute values before multiplication takes place. Integer operands may also be used. Independent multiplier operations are summarized in Table 11.

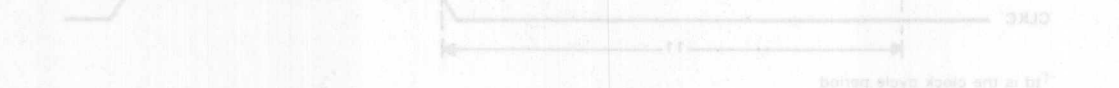


FIGURE 12. C REGISTER TIMING

The pipelined ALU contains a circuit for floating-point addition and/or subtraction of aligned operands. An exponent register, an exponent adjuster and a normalizer/rounder as shown in Figure 13. An exception circuit is provided to detect denormal inputs; these can be flushed to zero if the FAST input is set high. If the FAST input is low, the ALU accepts a denormal as input. A denormal exception flag (DENORM) goes high when the ALU output is a denormal.

Integer processing in the ALU includes both arithmetic and logical operations on either two's complement numbers or unsigned integers. The ALU performs addition, subtraction, comparison, logical shifts, logical AND, logical OR, and logical XOR.

If the operands to the multiplier are double precision or mixed precision (ie. one single precision and one double precision), then one extra clock cycle is required to get the product through the multiplier pipeline. This means that for PIPES1 = 1, one clock cycle is required for the multiplier pipeline; for PIPES1 = 0, two clock cycles are required for the multiplier pipeline.

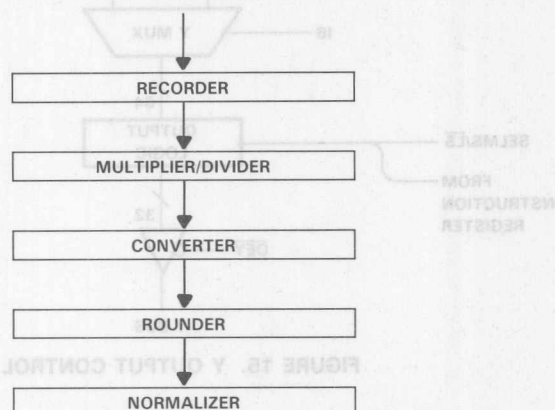


FIGURE 14. FUNCTIONAL DIAGRAM FOR MULTIPLIER

TABLE 11. INDEPENDENT MULTIPLIER OPERATIONS

| SINGLE OPERAND | TWO OPERANDS |
|----------------|--------------------|
| Square Root | Multiply Divide |

An exception circuit is provided to detect denormalized inputs; these are indicated by a high on the DENIN signal. Denormalized inputs must be wrapped by the ALU before multiplication, division, or square root. If results are wrapped (signaled by a high on the DENORM status pin), they must be unwrapped by the ALU.

The multiplier and ALU can be operated simultaneously by setting the I10 instruction input high. Division and square root are performed as independent multiplier operations, even though both multiplier and ALU are active during divide and SQRT operations.

data output controls

Selection and duration of results from the Y output multiplexer may be affected by several factors, including the operation selected, precision of the operands, registers enabled, and the next operation to be performed. The data output controls are not registered with the data and instruction inputs. When the device is microprogrammed, the effects of pipelining and sequencing of operations should be taken into account.

Two particular conditions need to be considered. Depending on which registers are enabled, an offset of one or more cycles must be allowed before a valid result is available at the Y output multiplexer. Also, certain sequences of operations may require both halves of a double-precision result to be read out within a single clock cycle. This is done by toggling the SELMS/ \overline{LS} signal in the middle of the clock period.

When a single-precision result is output, the SELMS/ \overline{LS} signal has no effect. The SELMS/ \overline{LS} signal is set low only to read out the LSH of a double-precision result (see Figure 15). To read out a result on the Y bus, the output enable OEY must be low. OEY is an asynchronous signal.

SN74ACT8847 64-BIT FLOATING-POINT UNIT

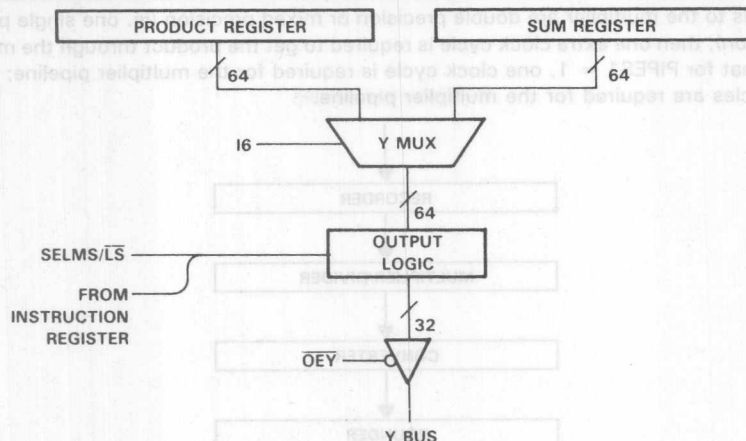


FIGURE 15. Y OUTPUT CONTROL

parity checker/generator

When BYTEP is high, internal even parity is generated for each byte of input data at the DA and DB ports and compared to the PA and PB parity inputs, respectively. If an odd number of bits is set high in a data byte, a parity check can also be performed on the entire input data word by setting BYTEP low. In this mode, PA0 is the parity input for DA data and PB0 is the parity input for DB data.

Even parity is generated for the Y multiplexer output, either for each byte or for each word of output, depending on the setting of BYTEP. When BYTEP is high, the parity generator computes four parity bits, one for each byte of the Y multiplexer output. Parity bits are output on the PY3-PY0 pins; PY0 represents parity for the least significant byte. A single parity bit can also be generated for the entire output data word by setting BYTEP low. In this mode, PY0 is the parity output.

master/slave comparator

A master/slave comparator is provided to compare data bytes from the Y output multiplexer and the status outputs with data bytes on the external Y and status ports when OEY, OES and OEC are high. If the data bytes are not equal, a high signal is generated on the master/slave error output pin (MSERR).

Figure 16 shows an example master/slave circuit. Two 'ACT8847 slave devices verify the data/status integrity of the 'ACT8847 master.

status and exception generation

A status and exception generator produces several output signals to indicate invalid operations as well as overflow, underflow, non-numerical and inexact results, in conformance with IEEE Std 754-1985. If output registers are enabled (PIPES2 = 0), status and exception results are latched in the status register on the rising edge of the clock. Status results are valid at the same time as associated data results are valid.

Duration and availability of status results are affected by the same timing constraints that apply to data results on the Y bus. Status outputs are enabled by two signals, OEC for comparison status and OES for other status and exception outputs. Status outputs are summarized in Tables 12 and 13.



TABLE 12. COMPARISON STATUS OUTPUTS

| SIGNAL | RESULT OF COMPARISON (ACTIVE HIGH) |
|--------|--|
| AEQB | The A and B operands are equal. A high signal on the AEQB output indicates a zero result from the selected source except during a compare operation in the ALU. During integer operations, indicates zero status output. |
| AGTB | The A operand is greater than the B operand. |
| UNORD | The two inputs of a comparison operation are unordered, i.e., one or both of the inputs is a NaN. |

During a compare operation in the ALU, the AEQB output goes high when the A and B operands are equal. When any operation other than a compare is performed, either by the ALU or the multiplier, the AEQB signal is used as a zero detect.



TABLE 13. STATUS OUTPUTS

| SIGNAL | STATUS RESULT |
|--------|---|
| CHEX | If I6 is low, indicates the multiplier is the source of an exception during a chained function. If I6 is high, indicates the ALU is the source of an exception during a chained function. |
| DENIN | Input to the multiplier is a denorm. When DENIN goes high, the STEX pins indicate which port had the denormal input. |
| DENORM | The multiplier output is a wrapped number or the ALU output is a denorm. In the FAST mode, this condition causes the result to go to zero. It also indicates an invalid integer operation, i.e., PASS (-A) with unsigned integer operand. |
| DIVBYO | An invalid operation involving a zero divisor has been detected by the multiplier. |
| ED | Exception detect status signal representing logical OR of all enabled exceptions in the exception disable register. |
| INEX | The result of an operation is not exact. |
| INF | The output is the IEEE representation of infinity. Not valid during float-to-integer conversions. |
| IVAL | A NaN has been input to the multiplier or the ALU, or an invalid operation $[(0 * \infty) \text{ or } (+\infty - \infty) \text{ or } (-\infty + \infty)]$ has been requested. This signal also goes high if an operation involves the square root of a negative number. When IVAL goes high, the STEX pins indicate which port had the NaN. |
| NEG | Output value has negative sign. |
| OVER | The result is greater than the largest allowable value for the specified format. |
| RNDCO | The mantissa of a number has been increased in magnitude by rounding. If the number generated was wrapped, then the unwrap round instruction must be used to properly unwrap the wrapped number (see Table 6). |
| SRCEX | The status was generated by the multiplier. (When SRCEX is low, the status was generated by the ALU.) |
| STEX0 | A NaN or a denorm has been input on the B port. |
| STEX1 | A NaN or a denorm has been input on the A port. |
| UNDER | The result is inexact and less than the minimum allowable value for the specified format. In the FAST mode, this condition causes the result to go to zero. |

In chained mode, results to be output are selected based on the state of the I6 (source output) pin (if I6 is low, ALU status will be selected; if I6 is high, multiplier status will be selected). If the nonselected output source generates an exception, CHEX is set high. Status of the nonselected output source can be forced using the SELST pins, as shown in Table 14.

TABLE 14. STATUS OUTPUT SELECTION (CHAINED MODE)

| SELST1- SELST0 | STATUS SELECTED |
|-------------------|---|
| 00 | Logical OR of ALU and multiplier exceptions (bit by bit) |
| 01 | Selects multiplier status |
| 10 | Selects ALU status |
| 11 | Normal operation (selection based on result source specified by I6 input) |

An exception detect mask register is available to mask out selected exceptions from the multiplier, ALU, or both. Multiply status is disabled during an independent ALU instruction, and ALU status is disabled during multiplier instructions. During chained operation, both status outputs are enabled.

When the exception mask register has been loaded with a mask, the mask is applied to the contents of the status register to disable unnecessary exceptions. Status results for enabled exceptions are then ORed together and, if true, the exception detect (ED) status output pin is set high (see Figure 18). Individual status outputs remain active and can be read independently from mask register operations.

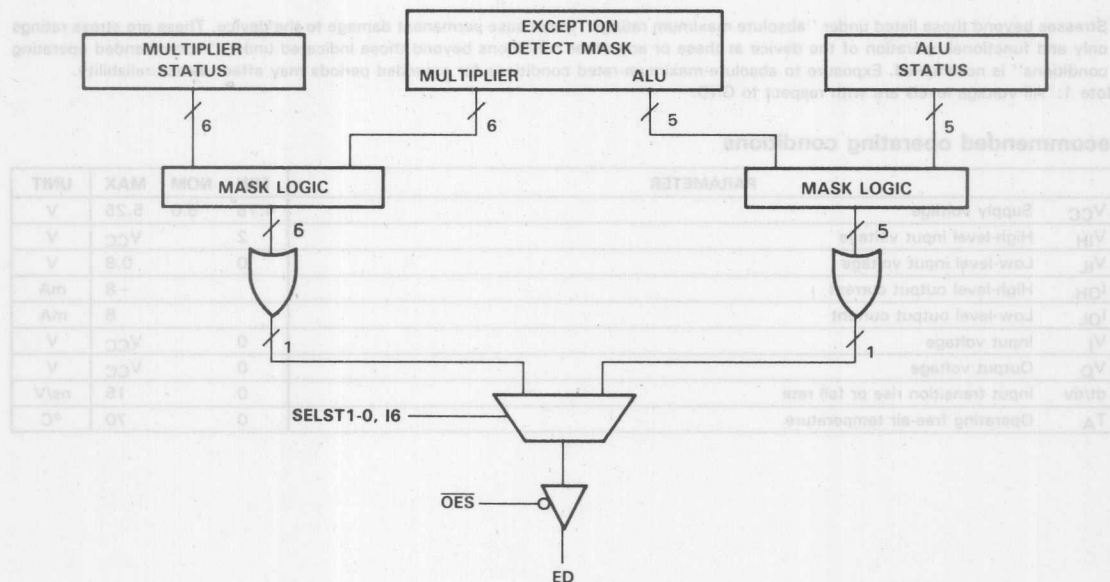


FIGURE 18. EXCEPTION DETECT MASK LOGIC

SN74ACT8847 **64-BIT FLOATING-POINT UNIT**

absolute maximum ratings over operating free-air temperature range (unless otherwise noted)[†]

| | |
|--|----------------|
| Supply voltage, V_{CC} (see Note 1) | −0.5 V to 6 V |
| Input clamp current, I_{IK} ($V_I < 0$ or $V_I > V_{CC}$) | ±20 mA |
| Output clamp current, I_{OK} ($V_O < 0$ or $V_O > V_{CC}$) | ±50 mA |
| Continuous output current, I_O ($V_O = V_{CC}$) | ±50 mA |
| Continuous current through V_{CC} or GND pins | ±100 mA |
| Operating free-air temperature range | 0°C to 70°C |
| Storage temperature range | −65°C to 150°C |

[†]Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

Note 1: All voltage levels are with respect to GND.

recommended operating conditions

| PARAMETER | | MIN | NOM | MAX | UNIT |
|-----------|------------------------------------|------|-----|----------|------|
| V_{CC} | Supply voltage | 4.75 | 5.0 | 5.25 | V |
| V_{IH} | High-level input voltage | 2 | | V_{CC} | V |
| V_{IL} | Low-level input voltage | 0 | | 0.8 | V |
| I_{OH} | High-level output current | | | −8 | mA |
| I_{OL} | Low-level output current | | | 8 | mA |
| V_I | Input voltage | 0 | | V_{CC} | V |
| V_O | Output voltage | 0 | | V_{CC} | V |
| dt/dv | Input transition rise or fall rate | 0 | | 15 | ns/V |
| T_A | Operating free-air temperature | 0 | | 70 | °C |

electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

| PARAMETER | TEST CONDITIONS | V _{CC} | T _A = 25°C | | | MIN | TYP | MAX | UNIT |
|------------------|---|-----------------|-----------------------|------|-----|------|-----|------|------|
| | | | MIN | TYP | MAX | | | | |
| V _{OH} | I _{OH} = -20 µA | 4.75 V | | | | 4.65 | | | V |
| | | 5.25 V | | | | 5.15 | | | |
| | I _{OH} = -8 mA | 4.75 V | | 3.95 | | 3.85 | | | |
| | | 5.25 V | | 4.7 | | 4.6 | | | |
| V _{OL} | I _{OL} = 20 µA | 4.75 V | | | | | | 0.1 | V |
| | | 5.25 V | | | | | | 0.1 | |
| | I _{OL} = 8 mA | 4.75 V | | 0.32 | | | | 0.45 | |
| | | 5.25 V | | 0.32 | | | | 0.45 | |
| I _I | V _I = V _{CC} or 0, 'ACT8847-30 | 5.25 V | | | | | | ±100 | µA |
| | V _I = V _{CC} or 0, 'ACT8847-40 | 5.25 V | | | | | | ±5 | |
| I _{OZ} | V _I = V _{CC} or 0, I _O 'ACT8847-30 | 5.25 V | | | | | | ±100 | µA |
| | V _I = V _{CC} or 0, I _O 'ACT8847-40 | 5.25 V | | | | | | ±10 | |
| I _{CCQ} | V _I = V _{CC} or 0, I _O 'ACT8847-30, 'ACT8847S-30 | 5.25 V | | | | | | 10 | mA |
| | V _I = V _{CC} or 0, I _O 'ACT8847-40 | 5.25 V | | | | | | 0.2 | |
| C _I | V _I = V _{CC} or 0 | 5 V | | 10 | | | | 10 | pF |

SN74ACT8847 **64-BIT FLOATING-POINT UNIT**

switching characteristics over recommended ranges of supply voltage and operating free-air temperature
(unless otherwise noted)

propagation delay times

| NO. | PARAMETER | FROM (INPUT) | TO (OUTPUT) | PIPELINE CONTROLS PIPES2-PIPES0 | SN74ACT8847-30 | | SN74ACT8847-40 | | UNIT |
|-----|------------|---|---------------------|---------------------------------------|----------------|----------------|----------------|----------------|------|
| | | | | | MIN | MAX | MIN | MAX | |
| 2 | t_{pd2} | INPUT REG | Y OUTPUT | 110 | | 66 | | 90 | ns |
| | | INPUT REG | STATUS | 110 | | 66 | | 90 | |
| 3 | t_{pd3} | PIPELN REG | Y OUTPUT | 10X | | 48 | | 60 | ns |
| | | PIPELN REG | STATUS | 10X | | 48 | | 60 | |
| 4 | t_{pd4} | OUTPUT REG | Y OUTPUT | 0XX | | 20 | | 24 | ns |
| | | OUTPUT REG | STATUS | 0XX | | 20 | | 24 | |
| 5 | t_{pd5} | SELMS/L \bar{S} | Y OUTPUT | XXX | | 13 | | 20 | ns |
| 6 | t_{pd6} | CLK \uparrow | Y OUTPUT INVALID | all but 111 | 4 | | 3 | | ns |
| 7 | t_{pd7} | CLK \uparrow | STATUS INVALID | all but 111 | 2 | | 3 | | ns |
| 8 | t_{pd8} | SELMS/L \bar{S} | Y OUTPUT INVALID | XXX | 1.5 | | 1.5 | | ns |
| 9 | t_{d1} | CLK \uparrow | CLK \uparrow | 010 w/o feedback | 56 | | 72 | | ns |
| | | CLK \uparrow | CLK \uparrow | 010 w/feedback [†] | 56 | | 72 | | |
| | | CLK \uparrow | CLK \uparrow | 010 W/FLOWC [‡] | 66 | | 84 | | |
| 10 | t_{d2} | CLK \uparrow | CLK \uparrow | 000 w/o feedback | 30 | | 40 | | |
| | | CLK \uparrow | CLK \uparrow | 000 w/feedback [†] | 30 | | 40 | | |
| | | CLK \uparrow | CLK \uparrow | 000 W/FLOWC [‡] | 40 | | 47 | | |
| 11 | t_{d3} | Delay time, CLKC after CLK to insure data captured in C register is data clocked into sum or product register by that clock. (PIPES2-PIPES0 = 0XX) | | | 14 | t_{d-0}^{\S} | 12 | t_{d-0}^{\S} | ns |
| 12 | t_{en} | \overline{OEY} | Y OUTPUT | XXX | | 14 | | 16 | ns |
| 13 | t_{en2} | $\overline{OEC}, \overline{OES}$ | STATUS | XXX | | 14 | | 16 | |
| 14 | t_{dis1} | \overline{OEY} | Y OUTPUT | XXX | | 14 | | 16 | |
| 15 | t_{dis2} | $\overline{OEC}, \overline{OES}$ | STATUS | XXX | | 14 | | 16 | |

[†]Applies to all cases except where operands are fed back using FLOWC to bypass C register.

[‡]Operands are fed back using FLOWC to bypass the C register.

[§] t_d is the clock cycle period.

SN74ACT8847
64-BIT FLOATING-POINT UNIT

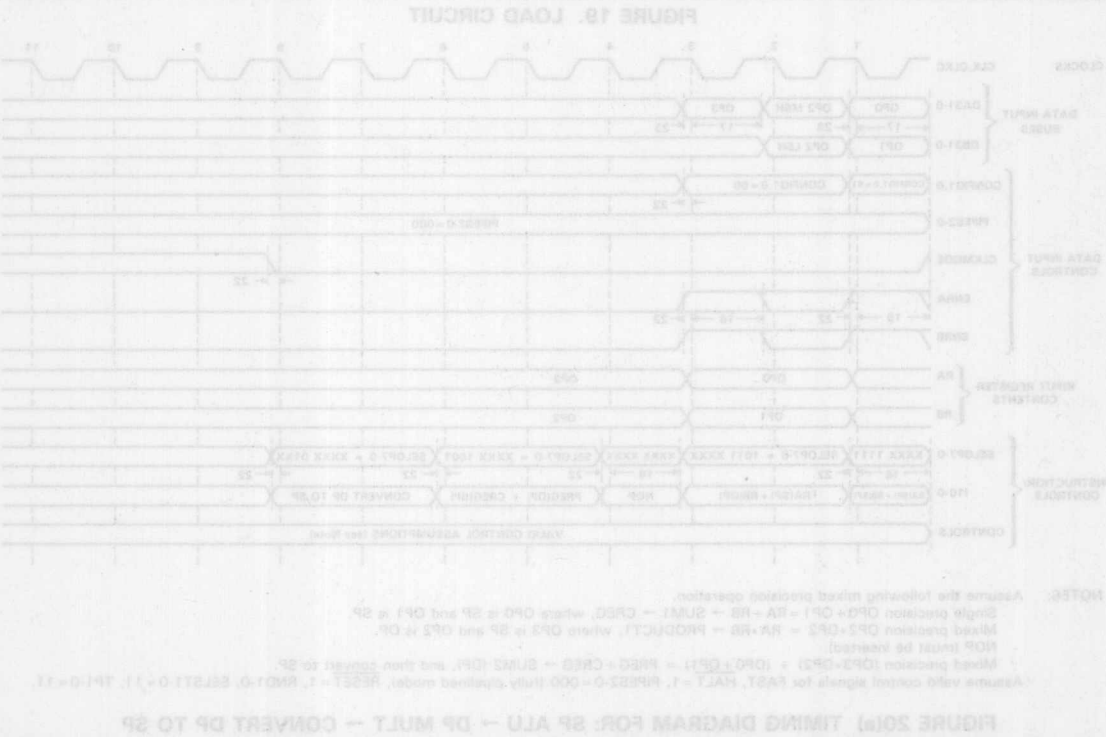
timing requirements over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

setup and hold times

| NO. | PARAMETER | | PIPELINE CONTROLS PIPES2-PIPE0 | SN74ACT8847-30 | | SN74ACT8847-40 | | UNIT |
|-----|-----------|----------------------------|-----------------------------------|----------------|-----|----------------|-----|------|
| | | | | MIN | MAX | MIN | MAX | |
| 16 | t_{su1} | Inst/control before CLK↑ | XX0 | 12 | | 14 | | ns |
| 17 | t_{su2} | DA/DB before CLK↑ | XX0 | 11 | | 13 | | |
| 18 | t_{su3} | DA/DB before 2nd CLK↑ (DP) | XX1 | 40 | | 52 | | |
| 19 | t_{su4} | CONFIG1-0 before CLK↑ | XX0 | 12 | | 14 | | |
| 20 | t_{su5} | SRCC before CLK↑ | XXX | 14 | | 14 | | |
| 21 | t_{su6} | RESET before CLK↑ | XX0 | 12 | | 14 | | |
| 22 | t_{h1} | Inst/control after CLK↑ | XXX | 3 | | 3 | | ns |
| 23 | t_{h2} | DA/DB after CLK↑ | XXX | 3 | | 3 | | |
| 24 | t_{h3} | SRCC after CLK↑ | XXX | 3 | | 3 | | |
| 25 | t_{h4} | RESET after CLK↑ | XX0 | 6 | | 6 | | |

CLK/RESET requirements

| PARAMETER | | SN74ACT8847-30 | | SN74ACT8847-40 | | UNIT |
|-----------|----------------|----------------|-----|----------------|-----|------|
| | | MIN | MAX | MIN | MAX | |
| t_w | Pulse duration | CLK high | 10 | 15 | | ns |
| | | CLK low | 10 | 15 | | |
| | | RESET | 10 | 12 | | |



SN74ACT8847 64-BIT FLOATING-POINT UNIT

PARAMETER MEASUREMENT INFORMATION

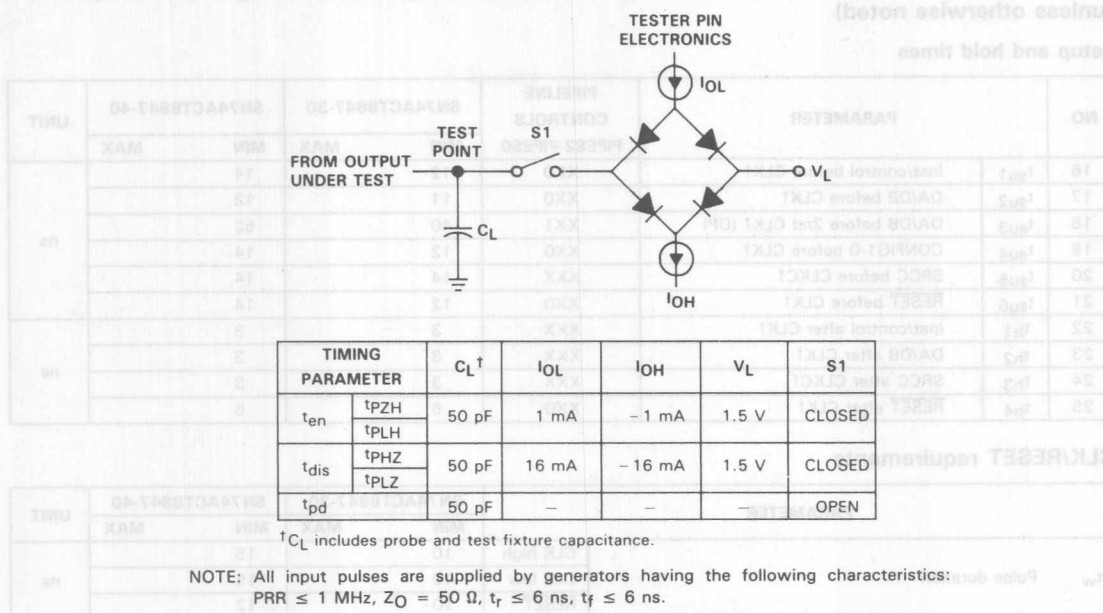
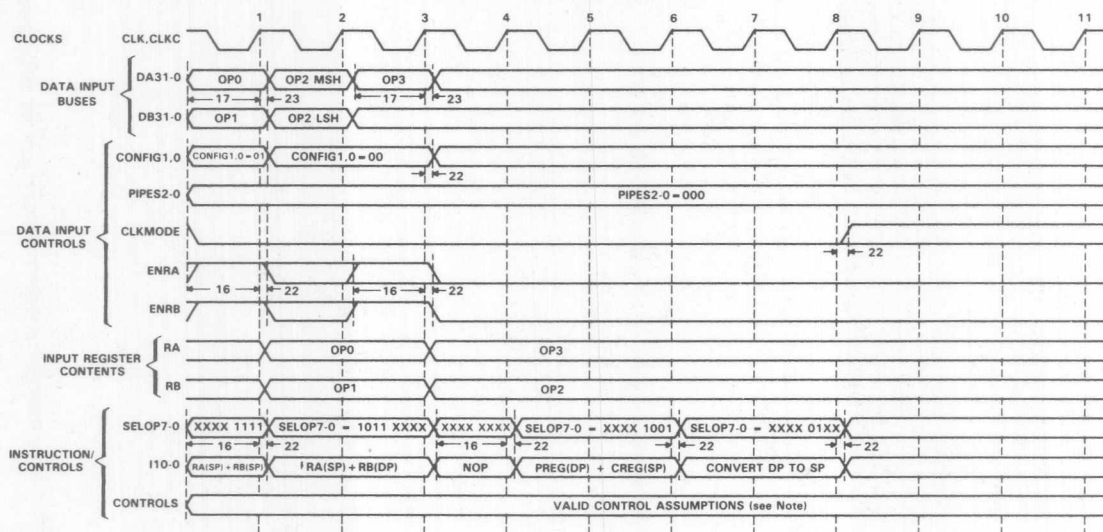


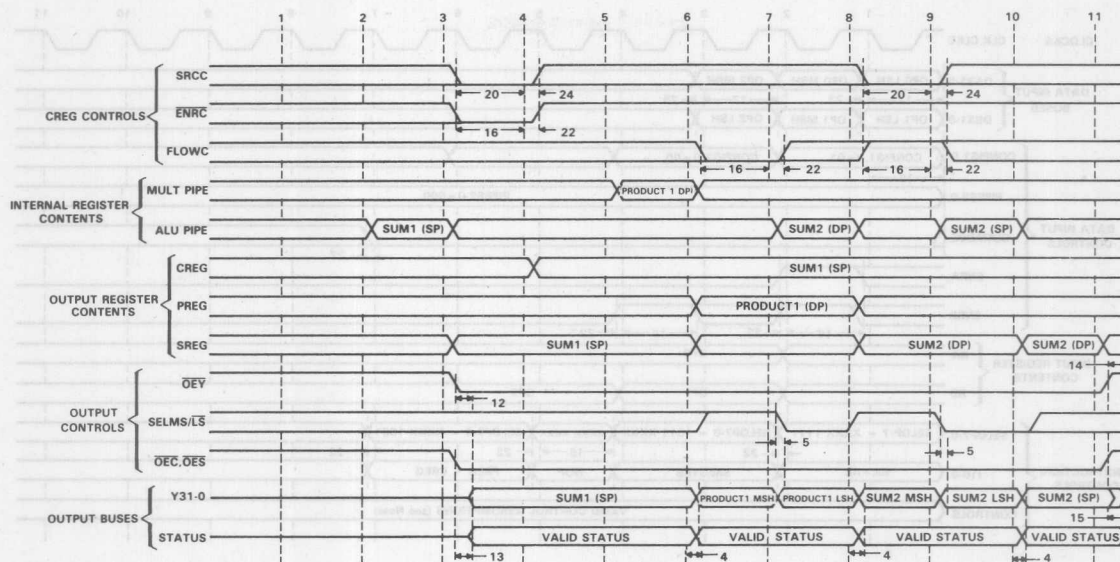
FIGURE 19. LOAD CIRCUIT



NOTES: Assume the following mixed precision operation.
 Single precision $OP0 + OP1 = RA + RB \rightarrow SUM1 \rightarrow CREG$, where $OP0$ is SP and $OP1$ is SP.
 Mixed precision $OP2 \cdot OP2 = RA \cdot RB \rightarrow PRODUCT1$, where $OP3$ is SP and $OP2$ is DP.
 NOP (must be inserted).
 Mixed precision $(OP3 \cdot OP2) + (OP0 + OP1) = PREG + CREG \rightarrow SUM2$ (DP), and then convert to SP.
 Assume valid control signals for FAST, HALT = 1, PIPES2-0 = 000 (fully pipelined mode), RESET = 1, RND1-0, SELST1-0 = 11, TP1-0 = 11.

FIGURE 20(a) TIMING DIAGRAM FOR: SP ALU → DP MULT → CONVERT DP TO SP

PARAMETER MEASUREMENT INFORMATION

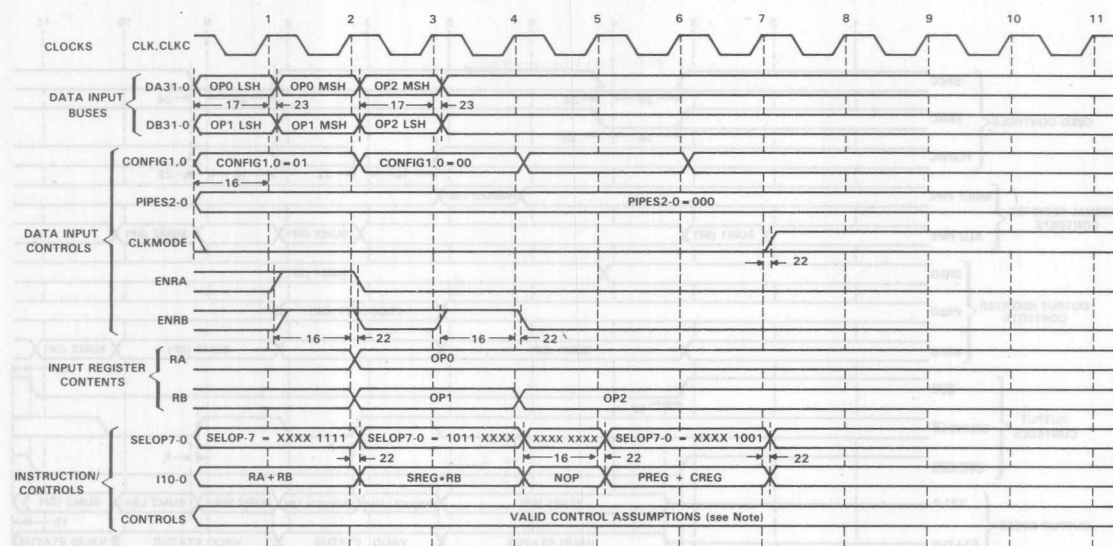


NOTES: Assume the following mixed precision operation.
Single precision $OP0 + OP1 = RA + RB \rightarrow SUM1 \rightarrow CREG$, where $OP0$ is SP and $OP1$ is SP.
Mixed precision $OP2 \cdot OP2 = RA \cdot RB \rightarrow PRODUCT1$, where $OP3$ is SP and $OP2$ is DP.
NOP (must be inserted).
Mixed precision $(OP3 \cdot OP2) + (OP0 + OP1) = PREG + CREG \rightarrow SUM2 (DP)$, and then convert to SP.
Assume valid control signals for FAST, HALT=1, PIPES2-0=000 (fully pipelined mode), RESET=1, RND1-0, SELST1-0=11, TP1-0=11.

FIGURE 20(b) TIMING DIAGRAM FOR: SP ALU \rightarrow DP MULT \rightarrow DP ALU \rightarrow CONVERT DP TO SP

SN74ACT8847 64-BIT FLOATING-POINT UNIT

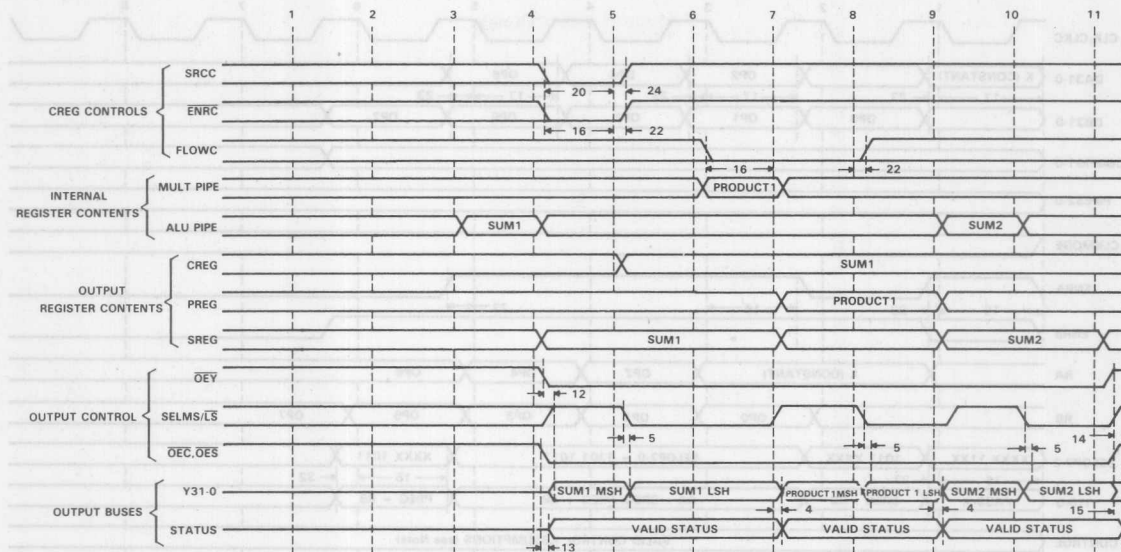
PARAMETER MEASUREMENT INFORMATION



NOTES: Assume the following double precision operation.
 $OP0 + OP1 = RA + RB \rightarrow SUM1 \rightarrow CREG$
 $(OP0 + OP1) * OP2 = SREG * RB \rightarrow PRODUCT1$
 $[(OP0 + OP1) * OP2] + (OP0 + OP1) = PREG + CREG \rightarrow SUM2$
 Assume valid control signals for FAST, HALT = 1, PIPES2-0 = 000 (fully pipelined model), RESET = 1, RND1-0, SELST1-0 = 11, TP1-0 = 11.

FIGURE 21(a) TIMING DIAGRAM FOR: DP ALU → DP MULT → DP ALU

PARAMETER MEASUREMENT INFORMATION

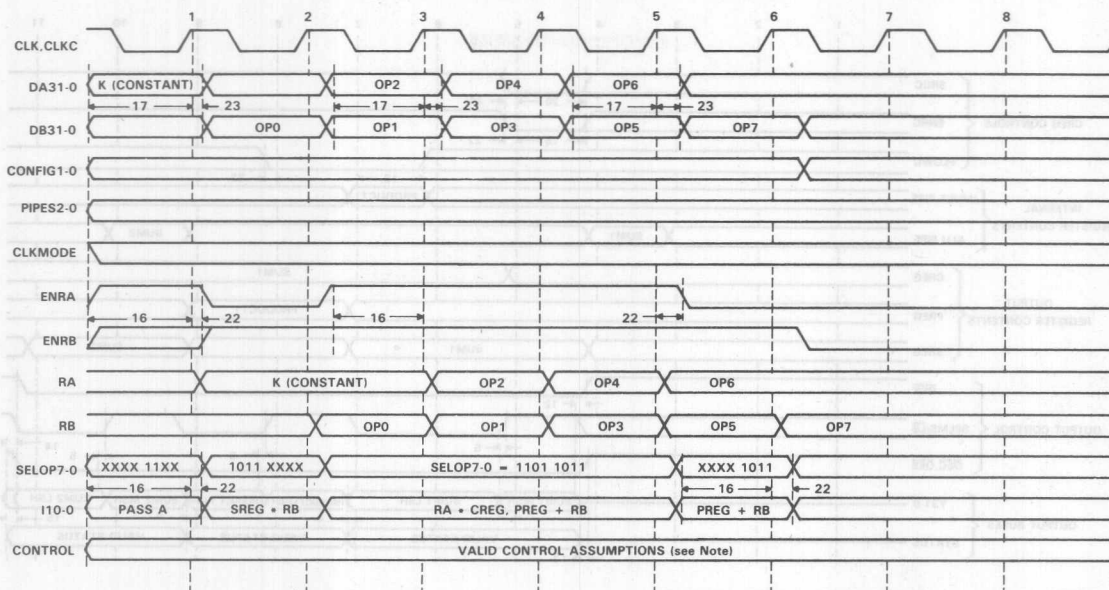


NOTES: Assume the following double precision operation.
 $OP0 + OP1 = RA + RB \rightarrow SUM1 \rightarrow CREG$
 $(OP0 + OP1) * OP2 = SREG * RB \rightarrow PRODUCT1$
 $[(OP0 + OP1) * OP2] + (OP0 + OP1) = PREG + CREG \rightarrow SUM2$
 Assume valid control signals for FAST, HALT=1, PIPES2-0=000 (fully pipelined mode), RESET=1, RND1-0, SELST1-0=11, TP1-0=11.

FIGURE 21(b) TIMING DIAGRAM FOR: DP ALU → DP MULT → DP ALU

SN74ACT8847 64-BIT FLOATING-POINT UNIT

PARAMETER MEASUREMENT INFORMATION



NOTES: Assume the following single precision operations.

(K * OP0) + OP1 = PRODUCT1 + OP1 → SUM1

(K * OP2) + OP3 = PRODUCT2 + OP3 → SUM2

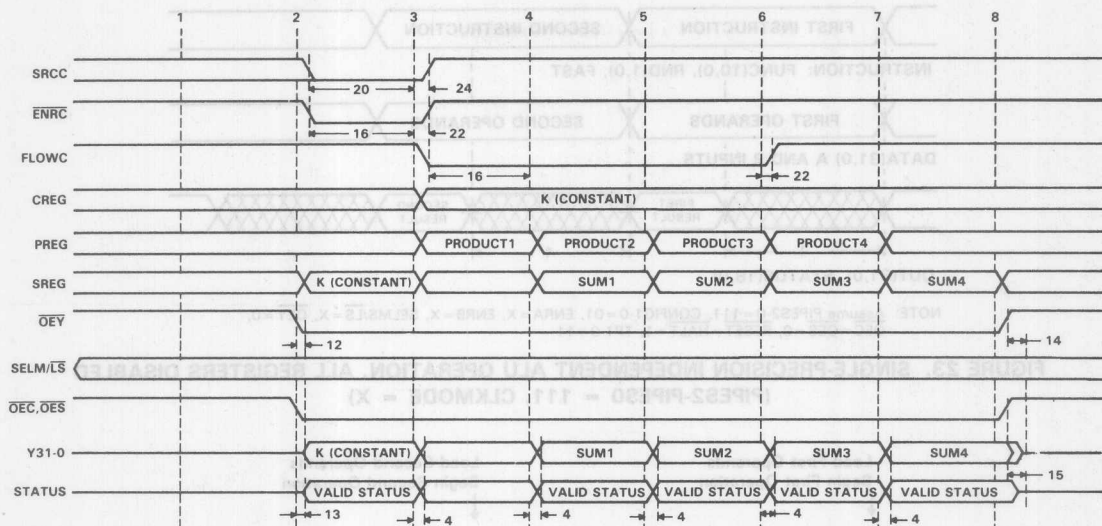
(K * OP4) + OP5 = PRODUCT3 + OP5 → SUM3

(K * OP6) + OP7 = PRODUCT4 + OP7 → SUM4

Assume valid control signals for FAST, HALT=1, PIPES2-0=010, RESET=1, RND1-0, SELST1-0=11, TP1-0=11.

FIGURE 22(a) TIMING DIAGRAM FOR: SP [(SCALAR * VECTOR) + VECTOR]

PARAMETER MEASUREMENT INFORMATION



NOTES: Assume the following single precision operations.

(K * OP0) + OP1 = PRODUCT1 + OP1 → SUM1

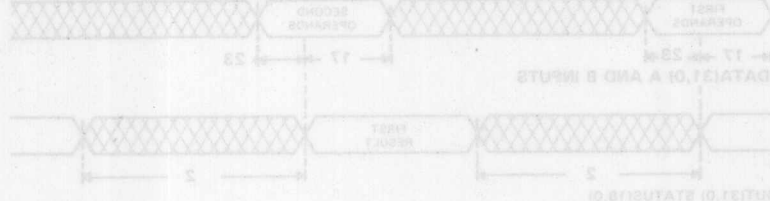
(K * OP2) + OP3 = PRODUCT2 + OP3 → SUM2

(K * OP4) + OP5 = PRODUCT3 + OP5 → SUM3

(K * OP6) + OP7 = PRODUCT4 + OP7 → SUM4

Assume valid control signals for FAST, HALT=1, PIPES2-0=010, RESET=1, RND1-0=11, SELST1-0=11, TP1-0=11.

FIGURE 22(b) TIMING DIAGRAM FOR: SP [(SCALAR * VECTOR) + VECTOR]



PARAMETER MEASUREMENT INFORMATION

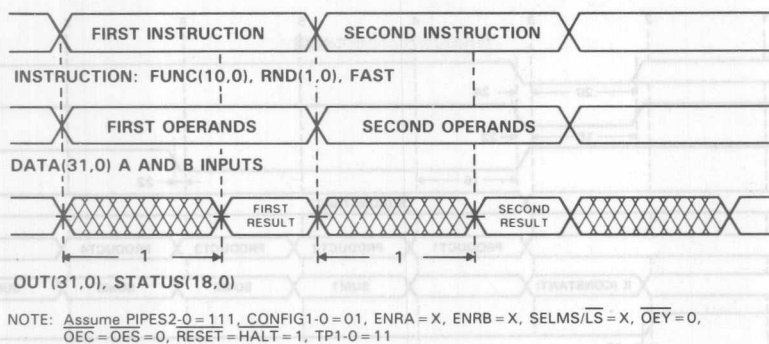


FIGURE 23. SINGLE-PRECISION INDEPENDENT ALU OPERATION, ALL REGISTERS DISABLED
(PIPES2-PIPES0 = 111, CLKMODE = X)

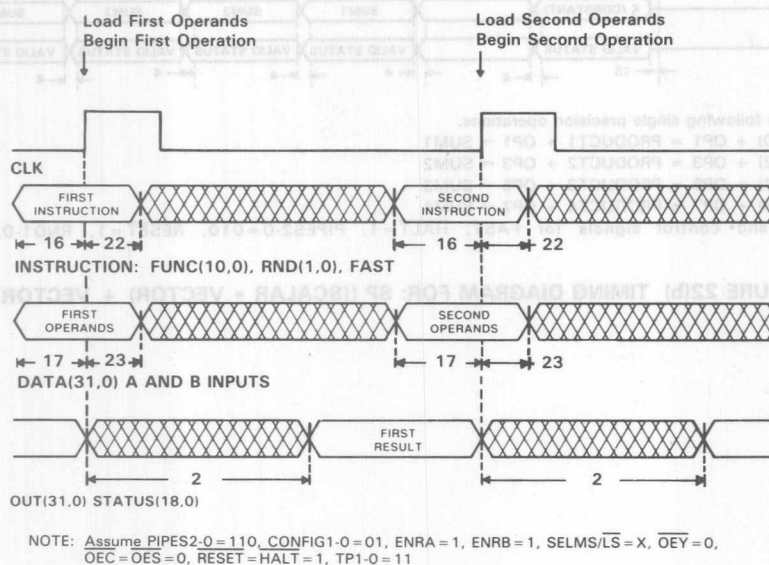


FIGURE 24. SINGLE-PRECISION INDEPENDENT ALU OPERATION, INPUT REGISTERS ENABLED
(PIPES2-PIPES0 = 110, CLKMODE = X)

PARAMETER MEASUREMENT INFORMATION

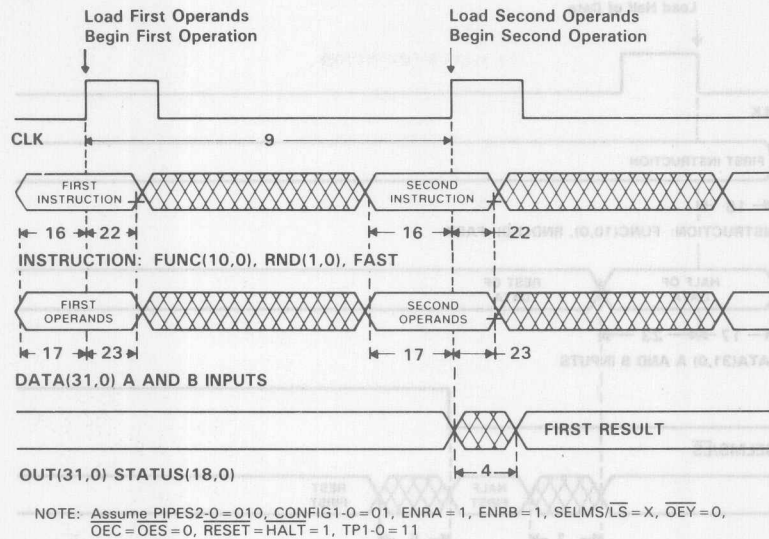


FIGURE 25. SINGLE-PRECISION INDEPENDENT ALU OPERATION, INPUT AND OUTPUT
REGISTERS ENABLED (PIPES2-PIPE0 = 010, CLKMODE = X)

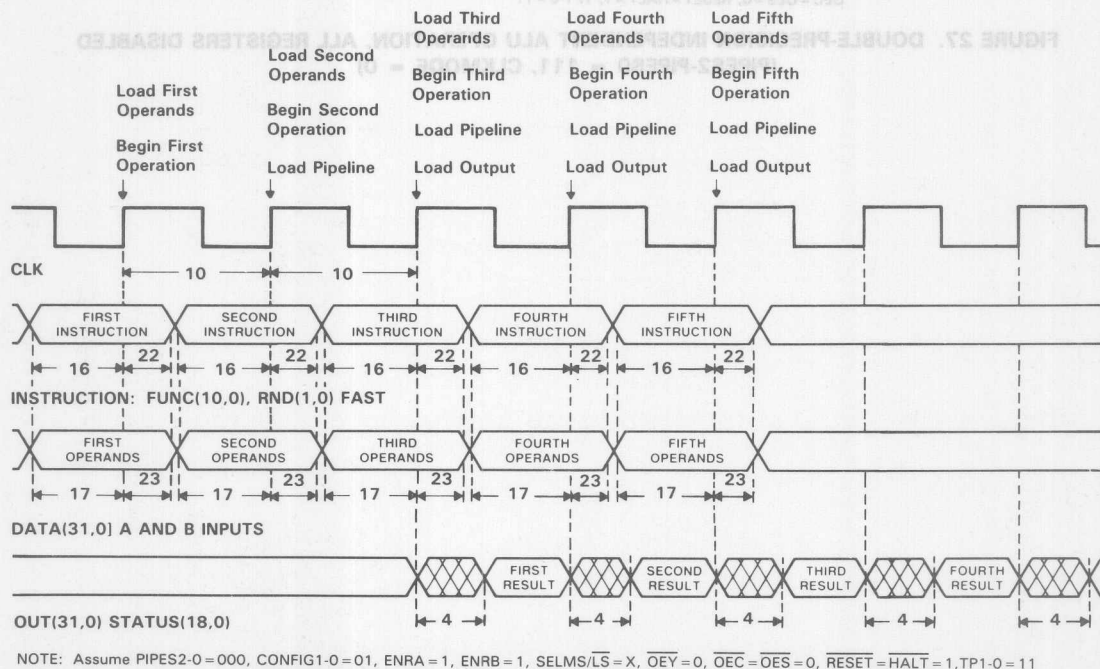
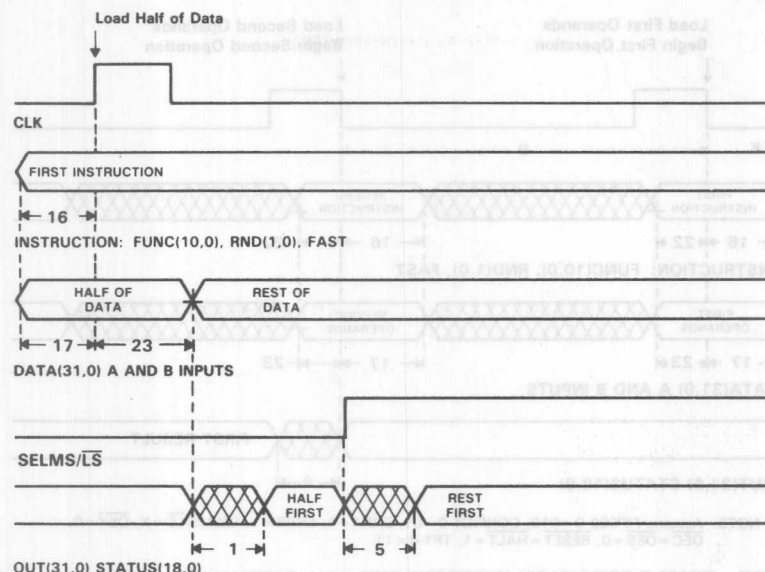


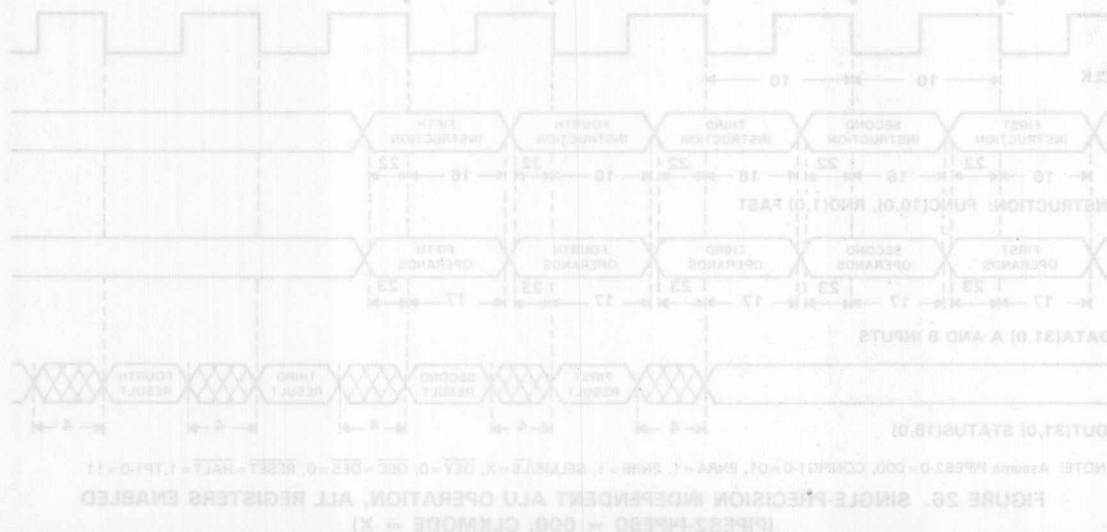
FIGURE 26. SINGLE-PRECISION INDEPENDENT ALU OPERATION, ALL REGISTERS ENABLED
(PIPES2-PIPE0 = 000, CLKMODE = X)

PARAMETER MEASUREMENT INFORMATION

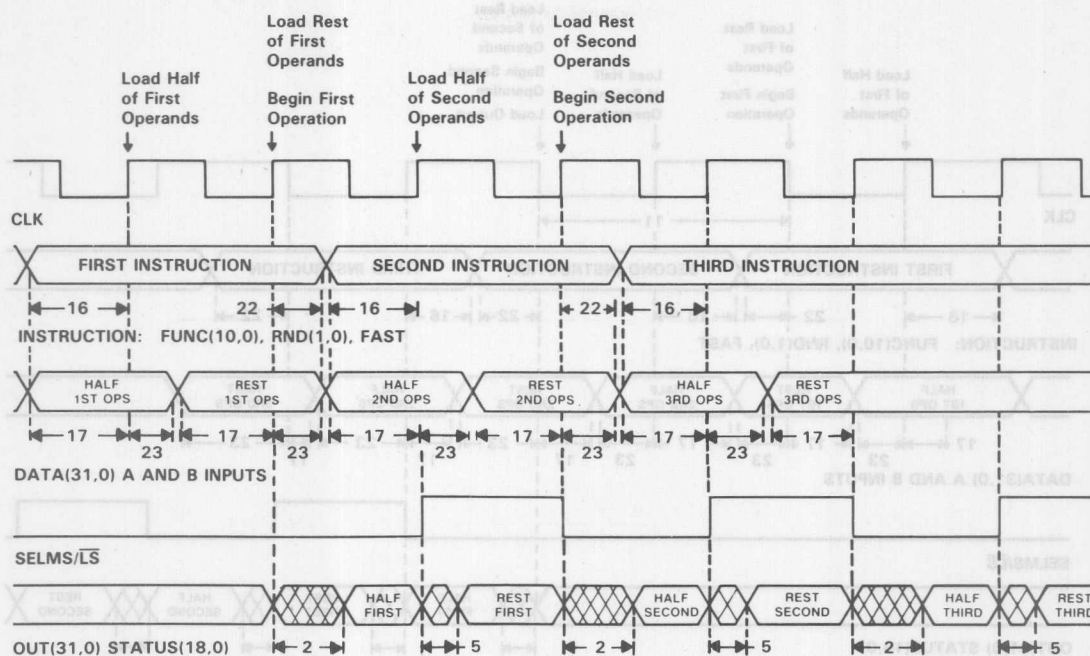


NOTE: Assume PIPES2-0 = 111, CLKMODE = 0, CONFIG1-0 = 11, ENRA = X, ENRB = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

FIGURE 27. DOUBLE-PRECISION INDEPENDENT ALU OPERATION, ALL REGISTERS DISABLED
(PIPES2-PIPES0 = 111, CLKMODE = 0)



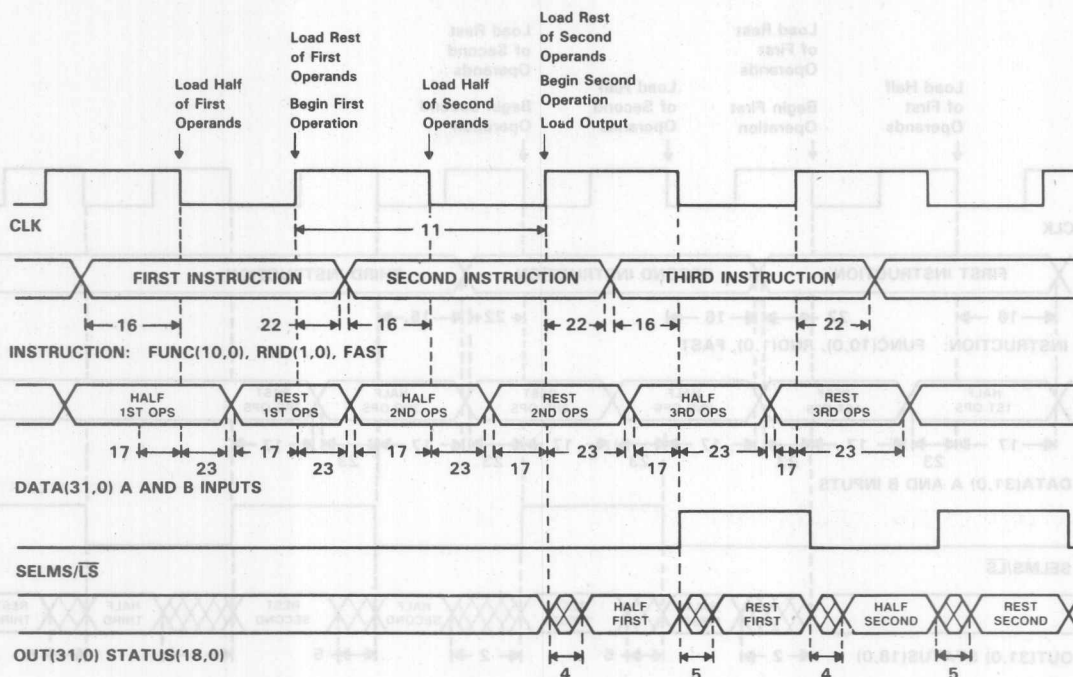
PARAMETER MEASUREMENT INFORMATION



NOTE: Assume PIPES2-0 = 110, CLKMODE = 0, CONFIG1-0 = 00, ENRA = 1, ENRB = 1, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

FIGURE 28. DOUBLE-PRECISION INDEPENDENT ALU OPERATION, INPUT REGISTERS ENABLED
(PIPES2-PIPES0 = 110, CLKMODE = 0)

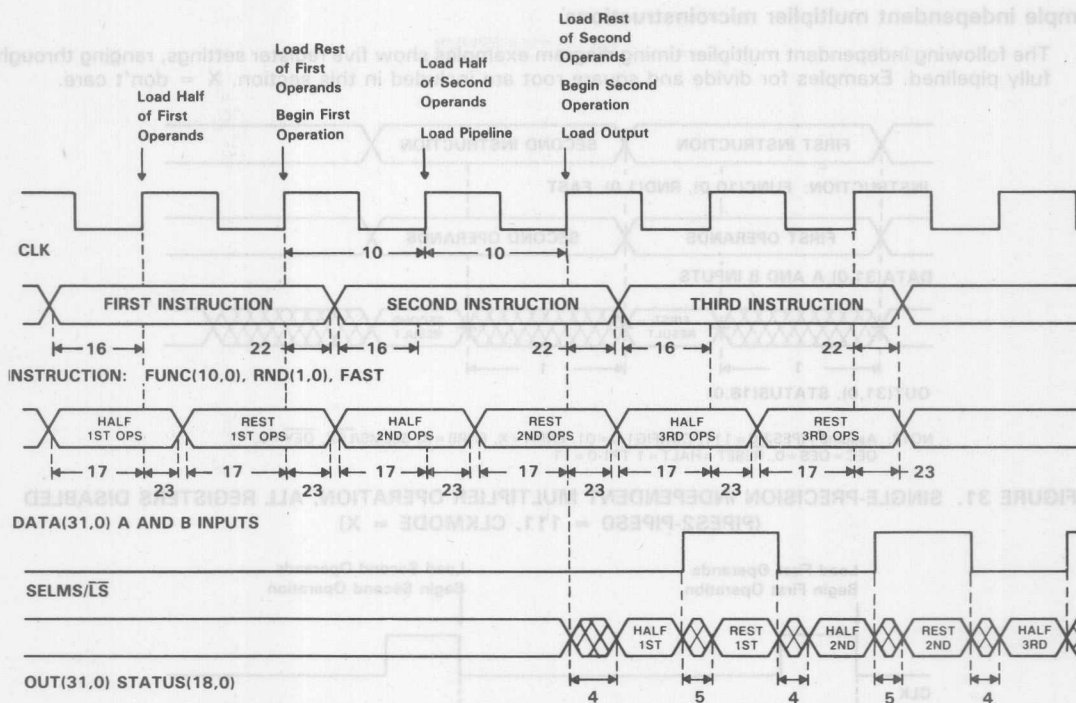
PARAMETER MEASUREMENT INFORMATION



NOTE: Assume PIPES2-0 = 010, CLKMODE = 1, CONFIG1-0 = 11, ENRA = 1, ENRB = 1, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

**FIGURE 29. DOUBLE-PRECISION INDEPENDENT ALU OPERATION, INPUT AND OUTPUT
 REGISTERS ENABLED (PIPES2-PIPE0 = 010, CLKMODE = 1)**

PARAMETER MEASUREMENT INFORMATION



NOTE: Assume PIPES2-0 = 000, CLKMODE = 0, CONFIG1-0 = 11, ENRA = 1, ENRB = 1, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

FIGURE 30. DOUBLE-PRECISION INDEPENDENT ALU OPERATION, ALL REGISTERS ENABLED
(PIPES2-PIPES0 = 000, CLKMODE = 0)

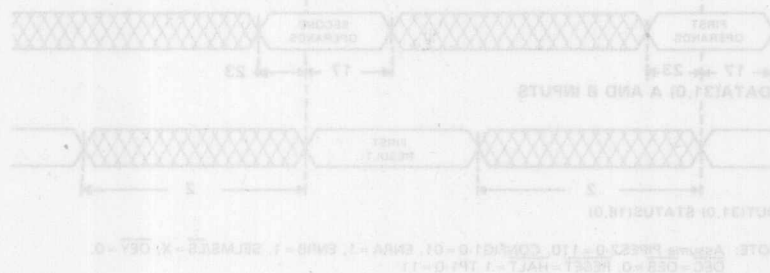
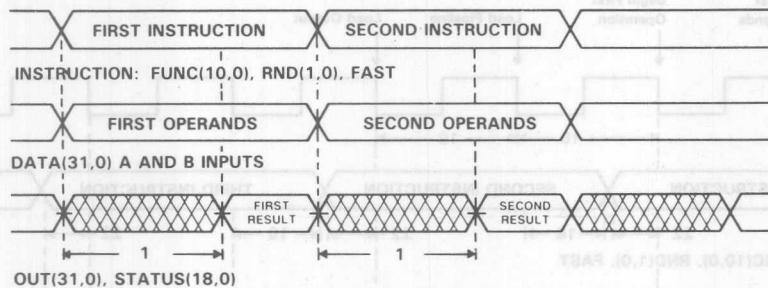


FIGURE 31. SINGLE-PRECISION INDEPENDENT MULTIPLIER OPERATION, ALL REGISTERS ENABLED
(PIPES2-PIPES0 = 010, CLKMODE = 0)

PARAMETER MEASUREMENT INFORMATION

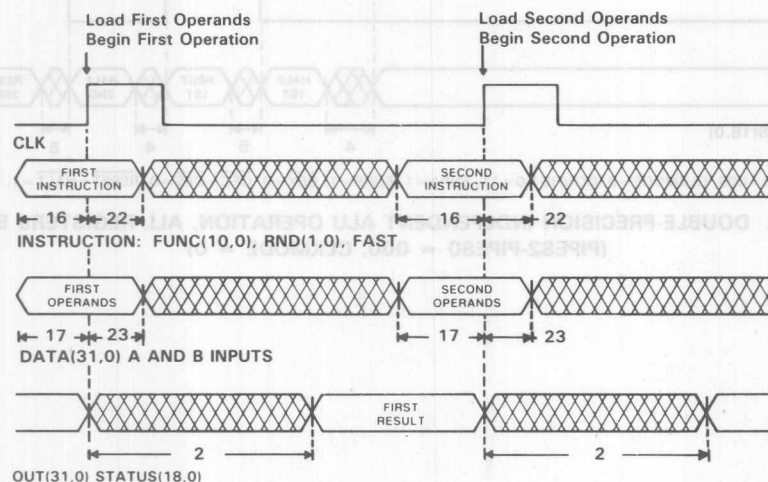
sample independent multiplier microinstructions

The following independent multiplier timing diagram examples show five register settings, ranging through fully pipelined. Examples for divide and square root are included in this section. X = don't care.



NOTE: Assume PIPES2-0 = 111, CONFIG1-0 = 01, ENRA = X, ENRB = X, SELMS/LSX, OEY = 0, OEC = OES = 0, RESET = HALT = 1 TP1-0 = 11

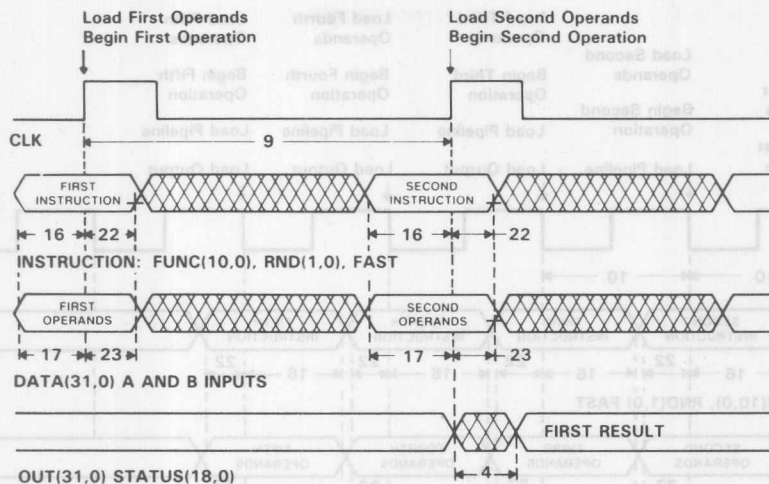
FIGURE 31. SINGLE-PRECISION INDEPENDENT MULTIPLIER OPERATION, ALL REGISTERS DISABLED (PIPES2-PIPES0 = 111, CLKMODE = X)



NOTE: Assume PIPES2-0 = 110, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/LSX, OEY = 0, OEC = OES = 0, RESET = HALT = 1 TP1-0 = 11

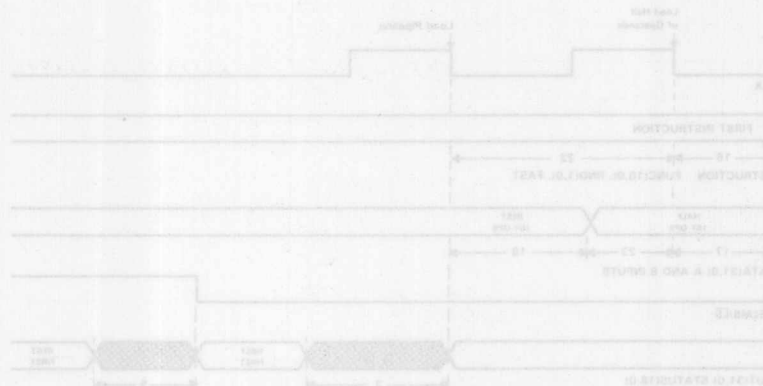
FIGURE 32. SINGLE-PRECISION INDEPENDENT MULTIPLIER OPERATION, INPUT REGISTERS ENABLED (PIPES2-PIPES0 = 010, CLKMODE = X)

PARAMETER MEASUREMENT INFORMATION



NOTE: Assume PIPES2-0=010, CONFIG1-0=01, ENRA=1, ENRB=1, SELMS/LS=X, OEY=0, OEC=OES=0, RESET=HALT=1 TP1-0=11

FIGURE 33. SINGLE-PRECISION INDEPENDENT MULTIPLIER OPERATION, INPUT AND OUTPUT
REGISTERS ENABLED (PIPES2-PIPE0 = 010, CLKMODE = X)

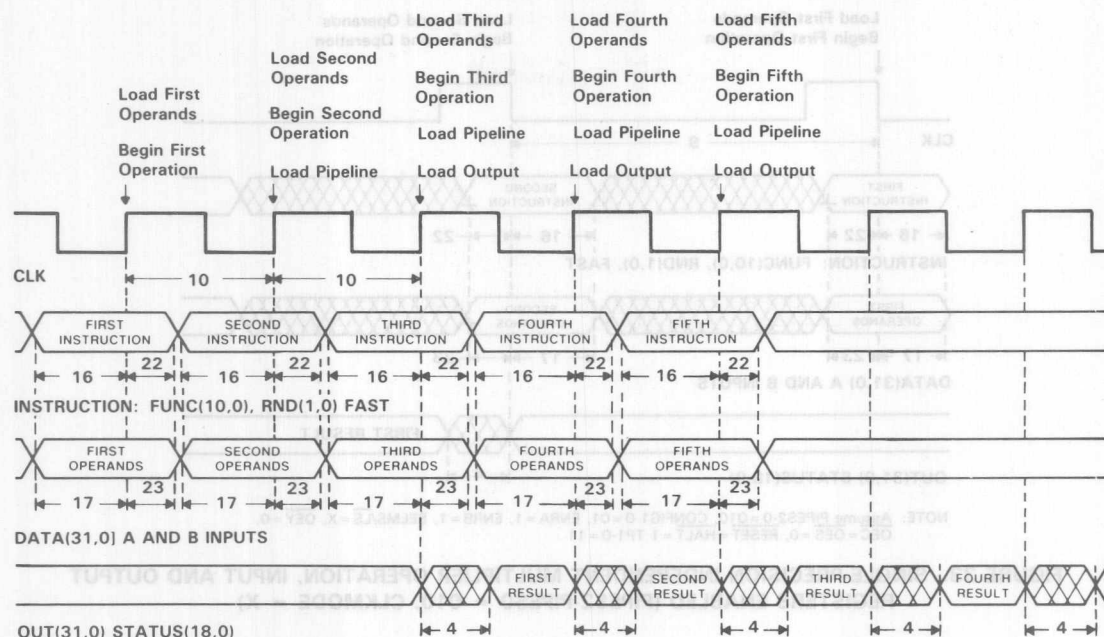


NOTE: Assume PIPES2-0 = 111, CLKMODE = 0, CONFIG1-0 = 11, ENRA = X, ENRB = X, SELMS/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

FIGURE 34. DOUBLE-PRECISION INDEPENDENT MULTIPLIER OPERATION, ALL REGISTERS DISABLED
(PIPES2-PIPE0 = 111, CLKMODE = 0)

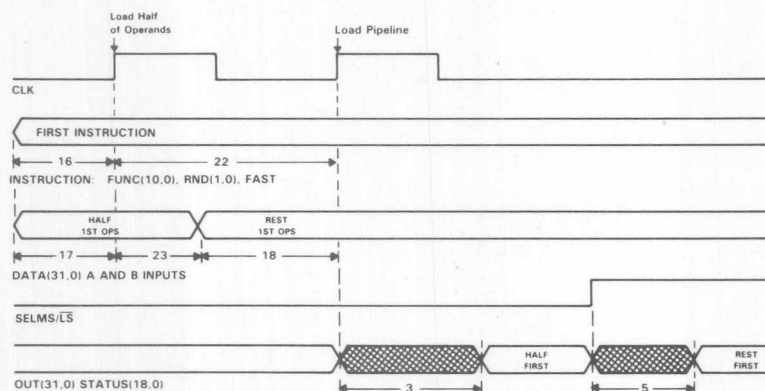
SN74ACT8847
64-BIT FLOATING-POINT UNIT

PARAMETER MEASUREMENT INFORMATION



NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

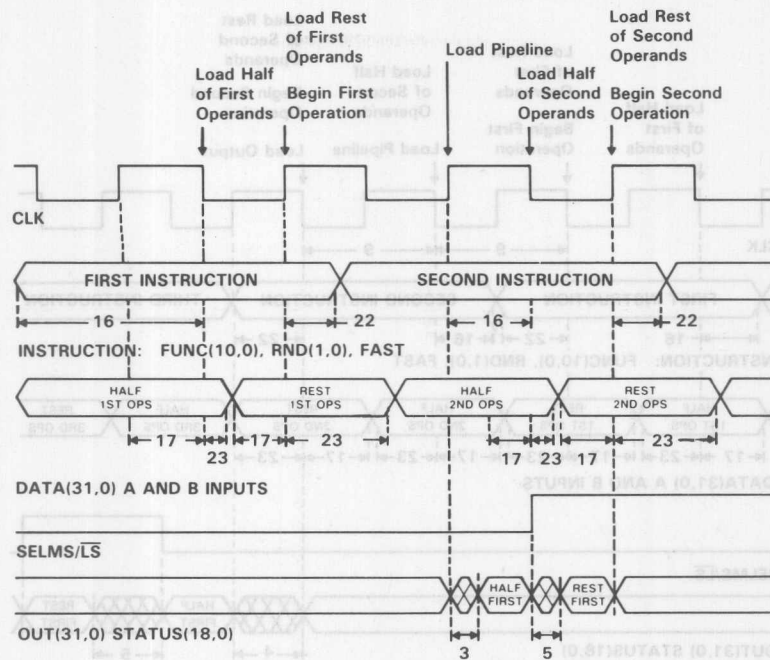
**FIGURE 34. SINGLE-PRECISION INDEPENDENT MULTIPLIER OPERATION, ALL REGISTERS ENABLED
(PIPES2-PIPES0 = 000, CLKMODE = X)**



NOTE: Assume PIPES2-0 = 111, CLKMODE = 0, CONFIG1-0 = 11, ENRA = X, ENRB = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

**FIGURE 35. DOUBLE-PRECISION INDEPENDENT MULTIPLIER OPERATION, ALL REGISTERS DISABLED
(PIPES2-PIPES0 = 111, CLKMODE = 0)**

PARAMETER MEASUREMENT INFORMATION



NOTE: Assume PIPES2-0 = 110, CONFIG1-0 = 11, ENRA = 1, ENRB = 1, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

FIGURE 36. DOUBLE-PRECISION INDEPENDENT MULTIPLIER OPERATION, INPUT REGISTERS ENABLED
(PIPES2-PIPES0 = 110, CLKMODE = 1)

PARAMETER MEASUREMENT INFORMATION

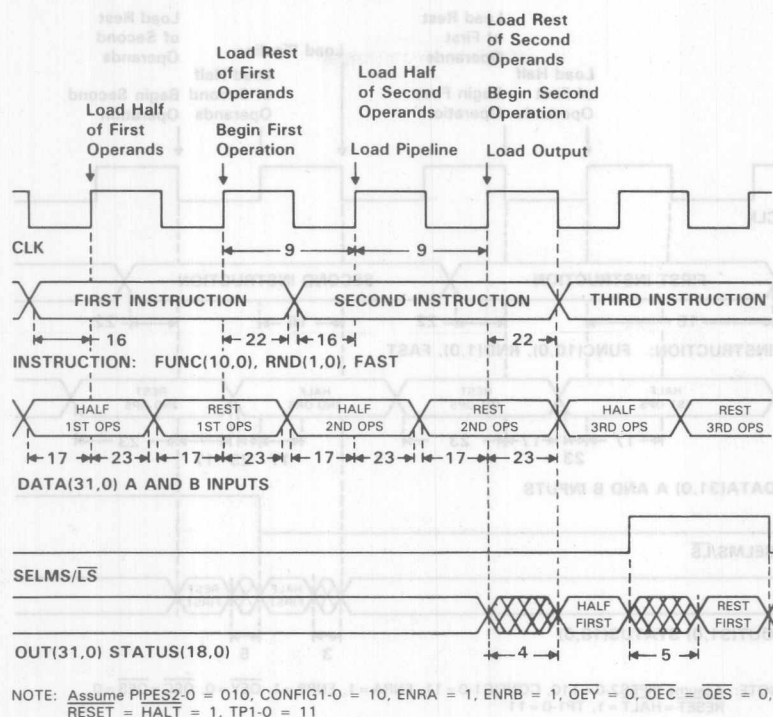
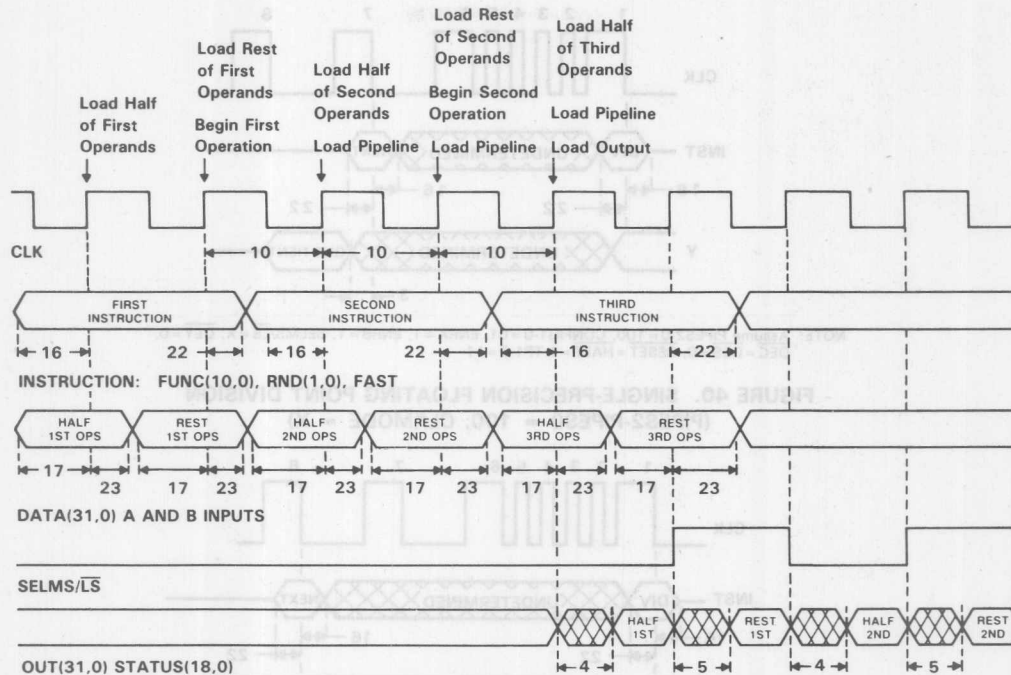


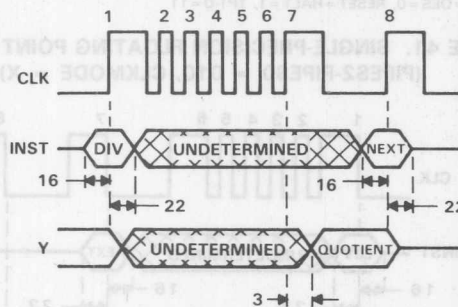
FIGURE 37. DOUBLE-PRECISION INDEPENDENT MULTIPLIER OPERATION, INPUT AND OUTPUT REGISTERS ENABLED (PIPES2-PIPE0 = 010, CLKMODE = 0)

PARAMETER MEASUREMENT INFORMATION



NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, OEY = 0, OEC = 0, OES = 0, RESET = HALT = 1, TP1-0 = 11

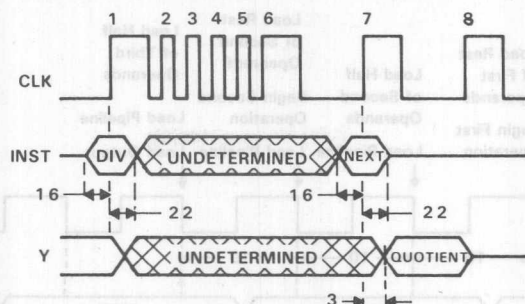
FIGURE 38. DOUBLE-PRECISION INDEPENDENT MULTIPLIER OPERATION, ALL REGISTERS ENABLED
(PIPES2-PIPE0 = 000, CLKMODE = 0)



NOTE: Assume PIPES2-0 = 110, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/L̄S = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

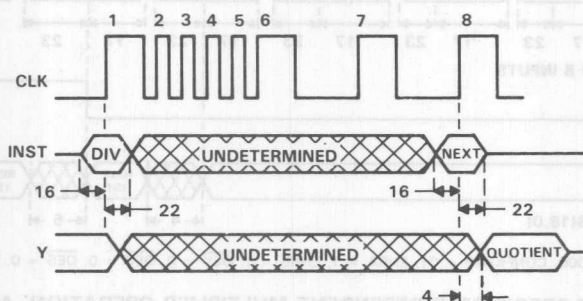
FIGURE 39. SINGLE-PRECISION FLOATING POINT DIVISION
(PIPES2-PIPE0 = 110, CLKMODE = X)

PARAMETER MEASUREMENT INFORMATION



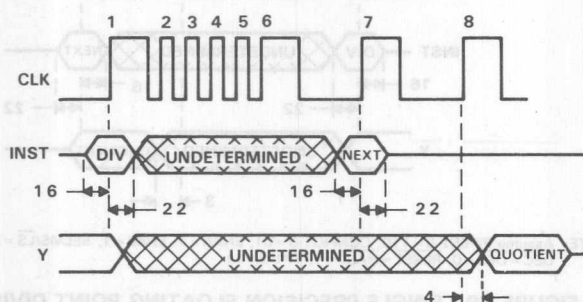
NOTE: Assume PIPES2-0 = 100, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1 TP1-0 = 11

FIGURE 40. SINGLE-PRECISION FLOATING POINT DIVISION
(PIPES2-PIPE0 = 100, CLKMODE = X)



NOTE: Assume PIPES2-0 = 010, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

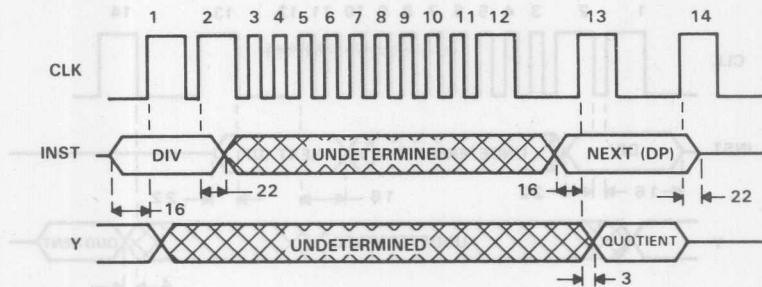
FIGURE 41. SINGLE-PRECISION FLOATING POINT DIVISION
(PIPES2-PIPE0 = 010, CLKMODE = X)



NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

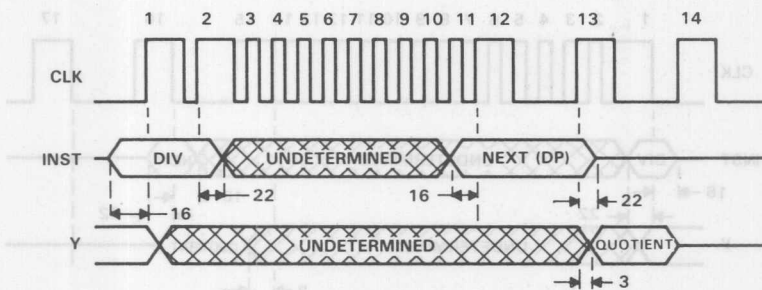
FIGURE 42. SINGLE-PRECISION FLOATING POINT DIVISION
(PIPES2-PIPE0 = 000, CLKMODE = X)

PARAMETER MEASUREMENT INFORMATION



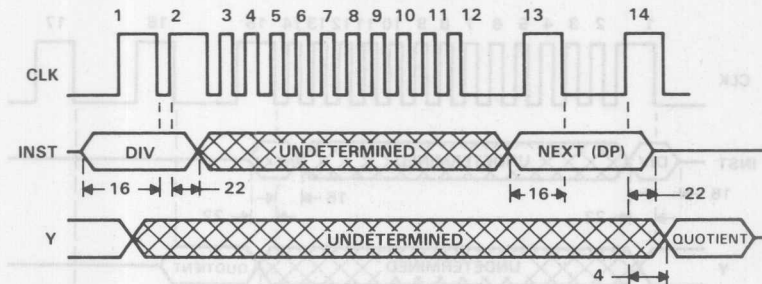
NOTE: Assume PIPES2-0 = 110, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, $\overline{OEY} = 0$, $\overline{OEC} = \overline{OES} = 0$, RESET = HALT = 1, TP1-0 = 11

FIGURE 43. DOUBLE-PRECISION FLOATING POINT DIVISION
(PIPES2-PIPE0 = 110, CLKMODE = 0)



NOTE: Assume PIPES2-0 = 100, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, $\overline{OEY} = 0$, $\overline{OEC} = \overline{OES} = 0$, RESET = HALT = 1, TP1-0 = 11

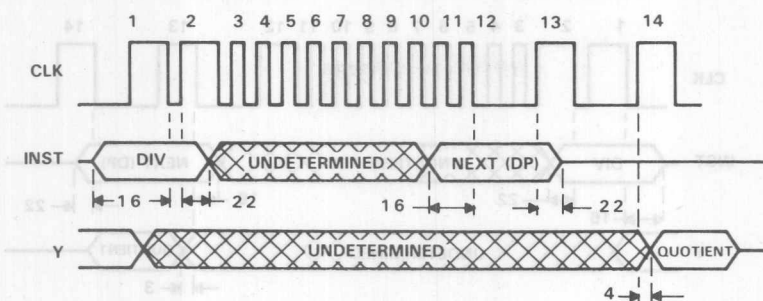
FIGURE 44. DOUBLE-PRECISION FLOATING POINT DIVISION
(PIPES2-PIPE0 = 100, CLKMODE = 0)



NOTE: Assume PIPES2-0 = 010, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/L \overline{S} = X, $\overline{OEY} = 0$, $\overline{OEC} = \overline{OES} = 0$, RESET = HALT = 1, TP1-0 = 11

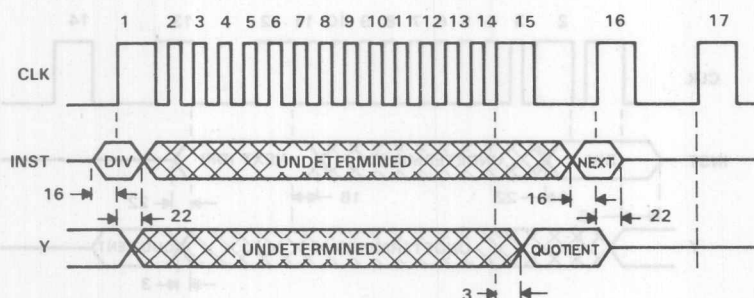
FIGURE 45. DOUBLE-PRECISION FLOATING POINT DIVISION
(PIPES2-PIPE0 = 010, CLKMODE = 1)

PARAMETER MEASUREMENT INFORMATION



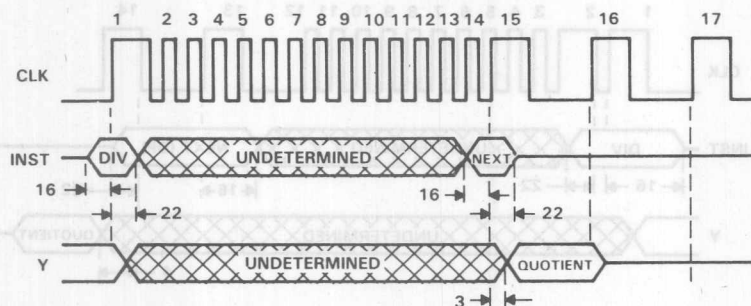
NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 00, ENRA = 1, ENRB = 1, $\overline{OEY} = 0$, $\overline{OEC} = \overline{OES} = 0$, RESET = HALT = 1, TP1-0 = 11.

FIGURE 46. DOUBLE-PRECISION FLOATING POINT DIVISION, ALL REGISTERS ENABLED
(PIPES2-PIPES0 = 000, CLKMODE = 1)



NOTE: Assume PIPES2-0 = 110, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/ \overline{LS} = X, $\overline{OEY} = 0$, $\overline{OEC} = \overline{OES} = 0$, RESET = HALT = 1, TP1-0 = 11. The result appears in the SREG.

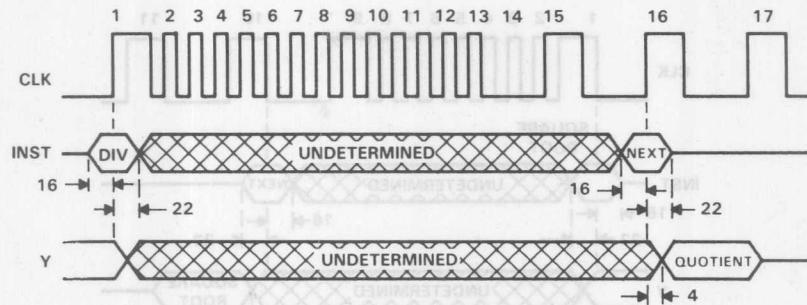
FIGURE 47. INTEGER DIVISION, INPUT REGISTERS ENABLED
(PIPES2-PIPES0 = 110, CLKMODE = X)



NOTE: Assume PIPES2-0 = 100, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/ \overline{LS} = X, $\overline{OEY} = 0$, $\overline{OEC} = \overline{OES} = 0$, RESET = HALT = 1, TP1-0 = 11. The result appears in the SREG.

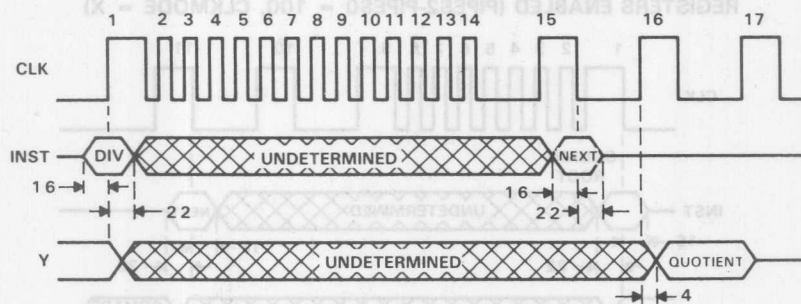
FIGURE 48. INTEGER DIVISION, INPUT AND PIPELINE REGISTERS ENABLED
(PIPES2-PIPES0 = 100, CLKMODE = X)

PARAMETER MEASUREMENT INFORMATION



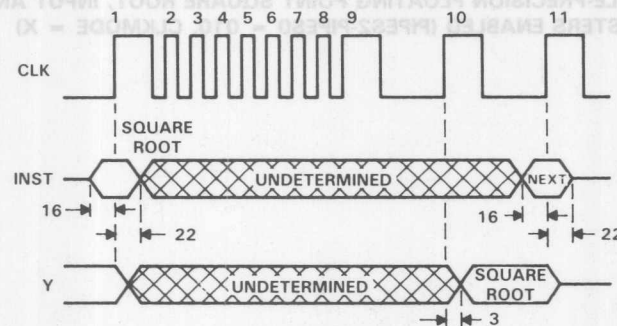
NOTE: Assume PIPES2-0 = 010, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11. The result appears in the SREG.

FIGURE 49. INTEGER DIVISION, INPUT AND OUTPUT REGISTERS ENABLED
(PIPES2-PIPES0 = 010, CLKMODE = X)



NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11. The result appears in the SREG.

FIGURE 50. INTEGER DIVISION, ALL REGISTERS ENABLED
(PIPES2-PIPES0 = 000, CLKMODE = X)



NOTE: Assume PIPES2-0 = 110, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

FIGURE 51. SINGLE-PRECISION FLOATING POINT SQUARE ROOT, INPUT REGISTERS ENABLED
(PIPES2-PIPES0 = 110, CLKMODE = X)

PARAMETER MEASUREMENT INFORMATION

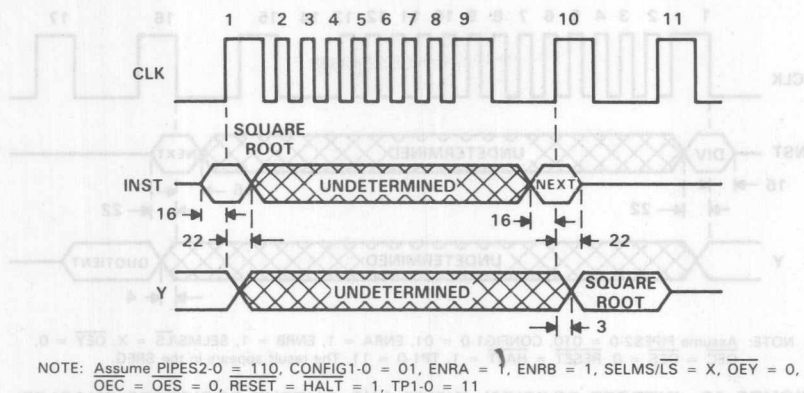


FIGURE 52. SINGLE-PRECISION FLOATING POINT SQUARE ROOT, INPUT AND PIPELINE REGISTERS ENABLED (PIPES2-PIPES0 = 100, CLKMODE = X)

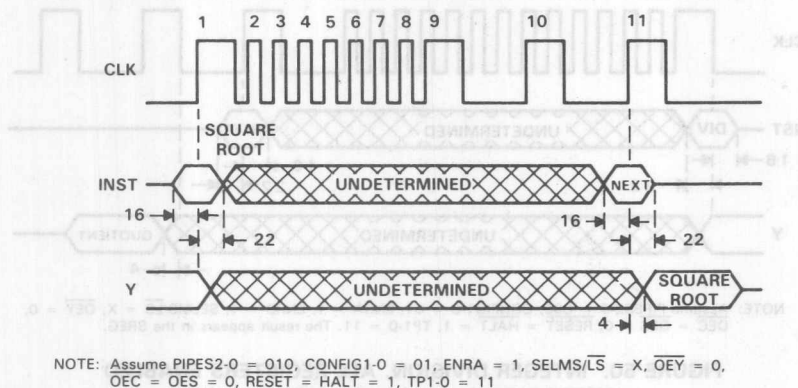


FIGURE 53. SINGLE-PRECISION FLOATING POINT SQUARE ROOT, INPUT AND OUTPUT REGISTERS ENABLED (PIPES2-PIPES0 = 010, CLKMODE = X)



FIGURE 54. SINGLE-PRECISION FLOATING POINT SQUARE ROOT, INPUT AND OUTPUT REGISTERS ENABLED (PIPES2-PIPES0 = 110, CLKMODE = X)

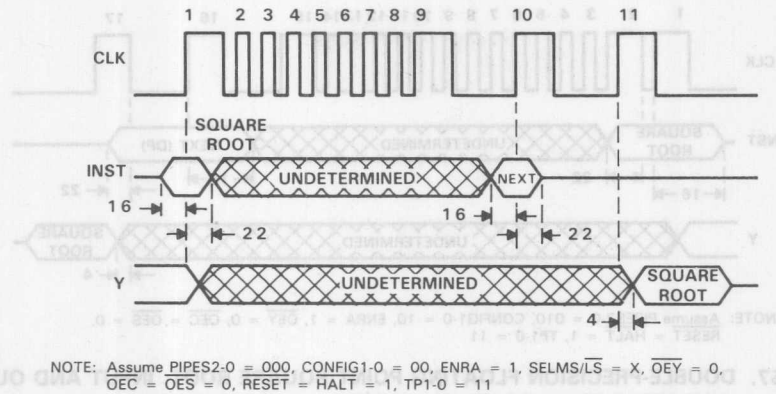


FIGURE 54. SINGLE-PRECISION FLOATING POINT SQUARE ROOT, ALL REGISTERS ENABLED
(PIPES2-PIPES0 = 000, CLKMODE = X)

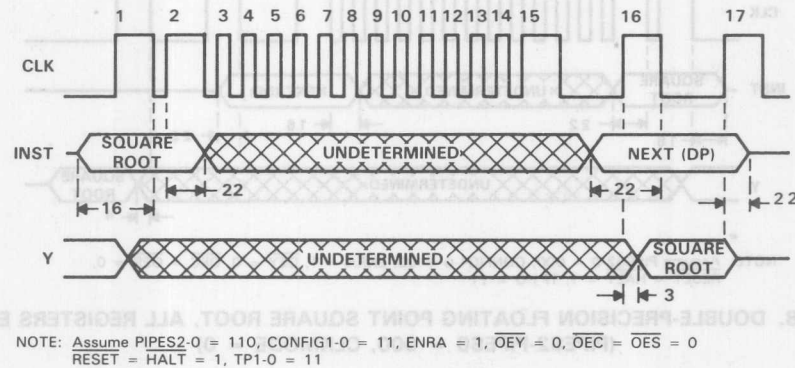


FIGURE 55. DOUBLE-PRECISION FLOATING POINT SQUARE ROOT, INPUT REGISTERS ENABLED (PIPES2-PIPES0 = 110, CLKMODE = 1)

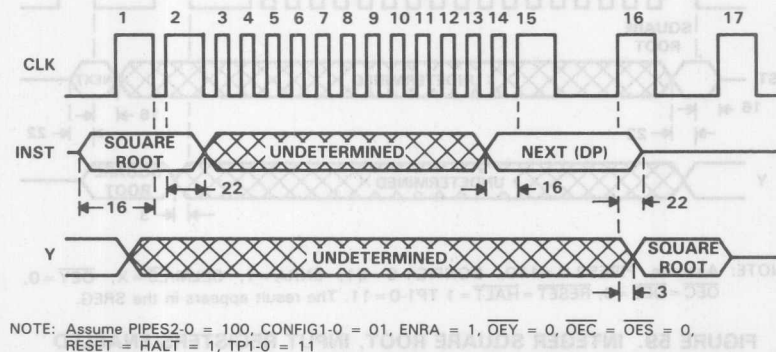
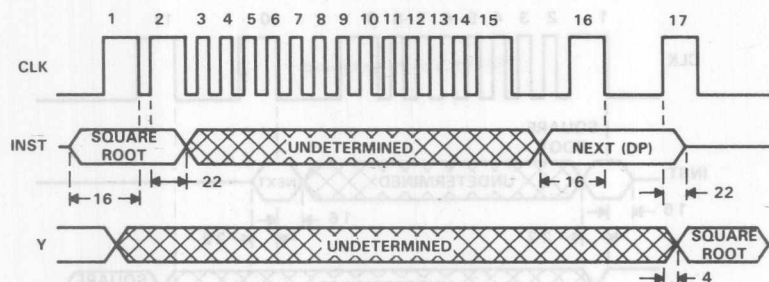


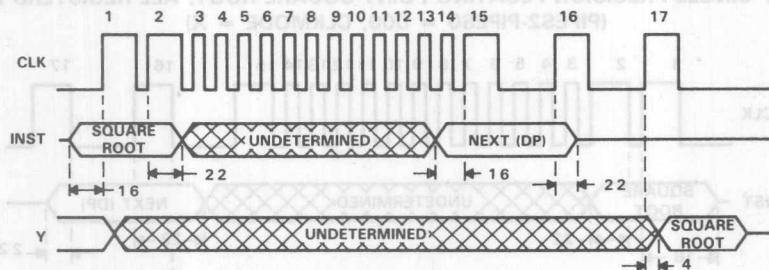
FIGURE 56. DOUBLE-PRECISION FLOATING POINT SQUARE ROOT, INPUT AND PIPELINE REGISTERS ENABLED (PIPES2-PIPES0 = 100, CLKMODE = 0)

PARAMETER MEASUREMENT INFORMATION



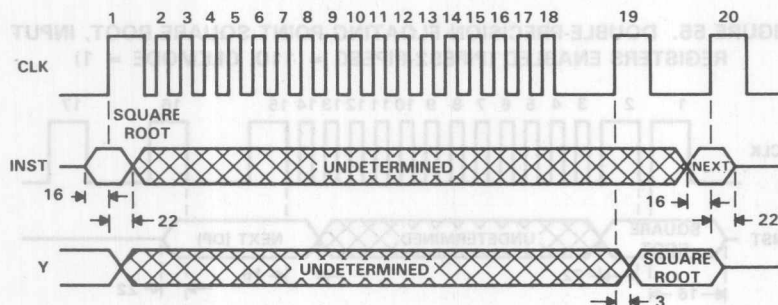
NOTE: Assume PIPES2-0 = 010, CONFIG1-0 = 10, ENRA = 1, $\overline{OEY} = 0$, $\overline{OEC} = \overline{OES} = 0$,
RESET = HALT = 1, TP1-0 = 11

FIGURE 57. DOUBLE-PRECISION FLOATING POINT SQUARE ROOT, INPUT AND OUTPUT
REGISTERS ENABLED (PIPES2-PIPES0 = 010, CLKMODE = 1)



NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 00, ENRA = 1, $\overline{OEY} = 0$, $\overline{OEC} = \overline{OES} = 0$,
RESET = HALT = 1, TP1-0 = 11

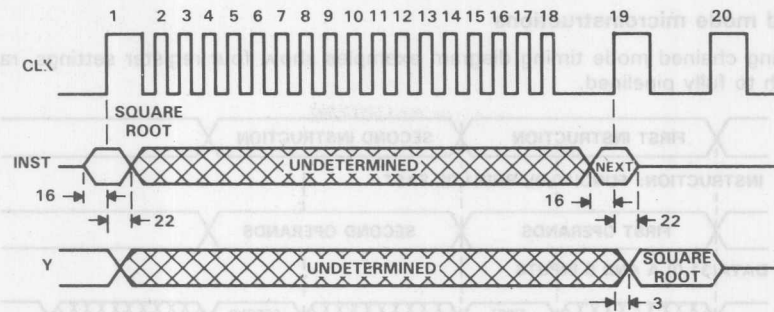
FIGURE 58. DOUBLE-PRECISION FLOATING POINT SQUARE ROOT, ALL REGISTERS ENABLED
(PIPES2-PIPES0 = 000, CLKMODE = 0)



NOTE: Assume PIPES2-0 = 110, CONFIG1-0 = 01, ENRA = 1, SELM/ \overline{LS} = X, $\overline{OEY} = 0$,
 $\overline{OEC} = \overline{OES} = 0$, RESET = HALT = 1 TP1-0 = 11. The result appears in the SREG.

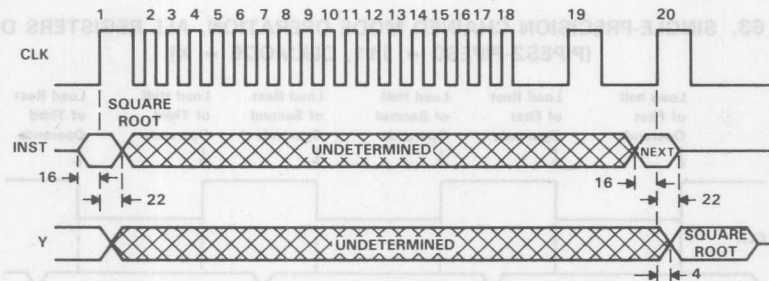
FIGURE 59. INTEGER SQUARE ROOT, INPUT REGISTERS ENABLED
(PIPES2-PIPES0 = 110, CLKMODE = X)

PARAMETER MEASUREMENT INFORMATION



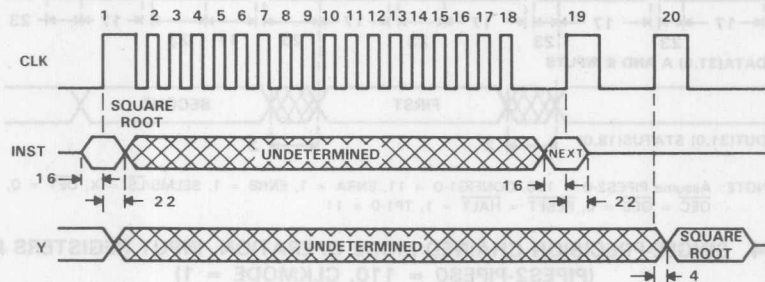
NOTE: Assume PIPES2-0 = 100, CONFIG1-0 = 00, ENRA = 1, SELMS/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11. The result appears in the SREG.

FIGURE 60. INTEGER SQUARE ROOT, INPUT AND PIPELINE REGISTERS ENABLED
(PIPES2-PIPE0 = 100, CLKMODE = X)



NOTE: Assume PIPES2-0 = 010, CONFIG1-0 = 01, ENRA = 1, SELMS/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11. The result appears in the SREG.

FIGURE 61. INTEGER SQUARE ROOT, INPUT AND OUTPUT REGISTERS ENABLED
(PIPES2-PIPE0 = 010, CLKMODE = X)



NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 00, ENRA = 1, SELMS/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11. The result appears in the SREG.

FIGURE 62. INTEGER SQUARE ROOT, ALL REGISTERS ENABLED
(PIPES2-PIPE0 = 000, CLKMODE = X)

PARAMETER MEASUREMENT INFORMATION

sample chained mode microinstructions

The following chained mode timing diagram examples show four register settings, ranging from fully flowthrough to fully pipelined.

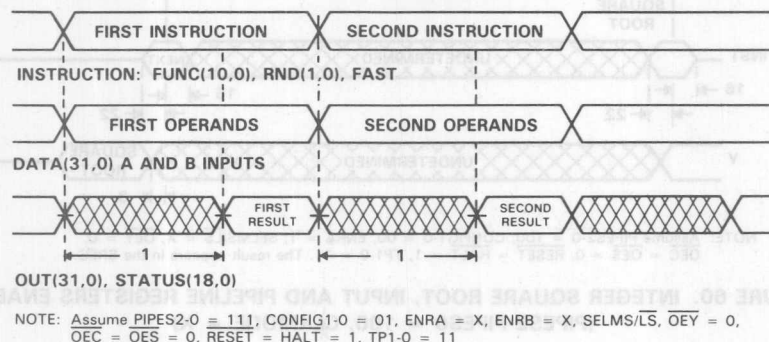


FIGURE 63. SINGLE-PRECISION CHAINED MODE OPERATION, ALL REGISTERS DISABLED
(PIPES2-PIPES0 = 111, CLKMODE = X)

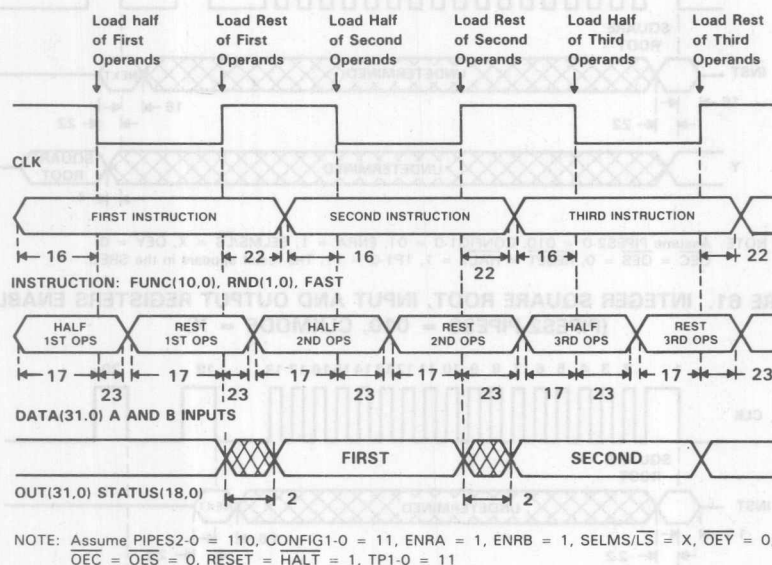
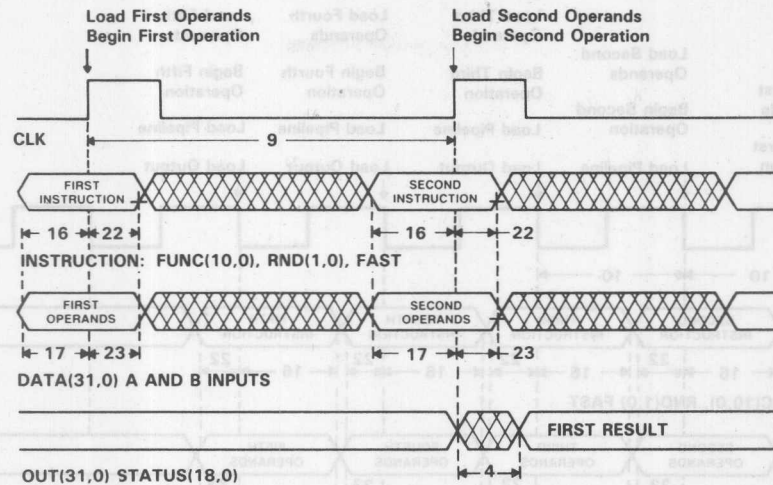


FIGURE 64. SINGLE-PRECISION CHAINED MODE OPERATION, INPUT REGISTERS ENABLED
(PIPES2-PIPES0 = 110, CLKMODE = 1)

PARAMETER MEASUREMENT INFORMATION



NOTE: Assume PIPES2-0 = 010, CONFIG1-0 = 01, ENRA = 1, SELMS/LS = X, OEY = 0,
OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

FIGURE 65. SINGLE-PRECISION CHAINED MODE OPERATION, INPUT AND OUTPUT REGISTERS
ENABLED (PIPES2-PIPES0 = 010, CLKMODE = X)

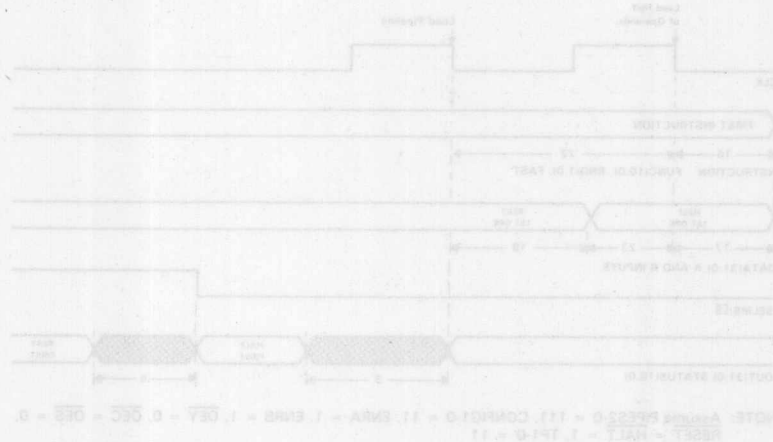
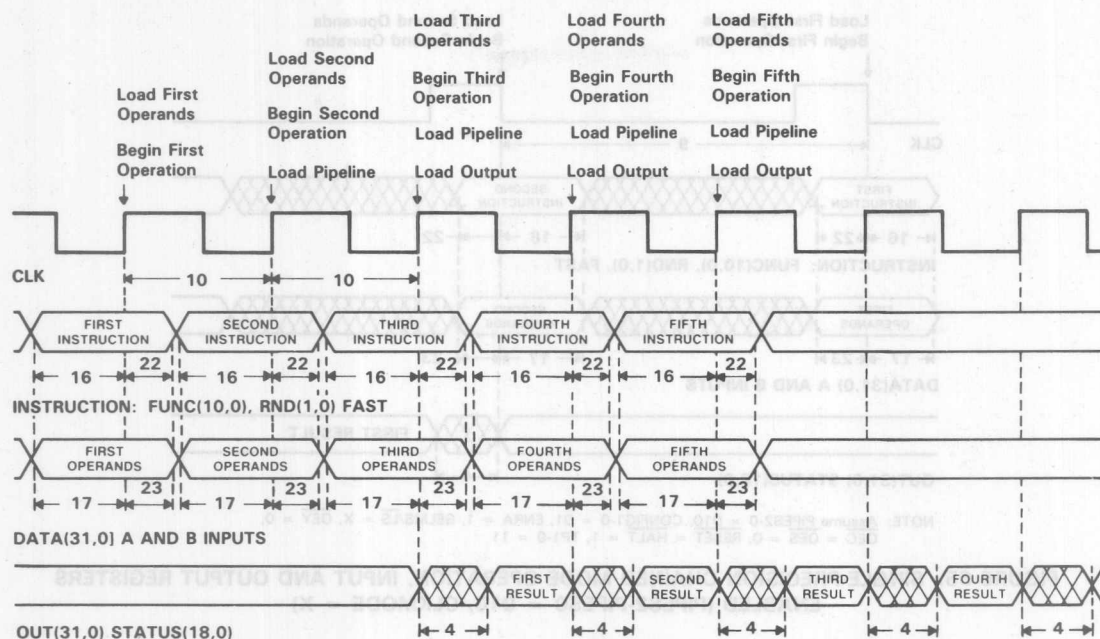


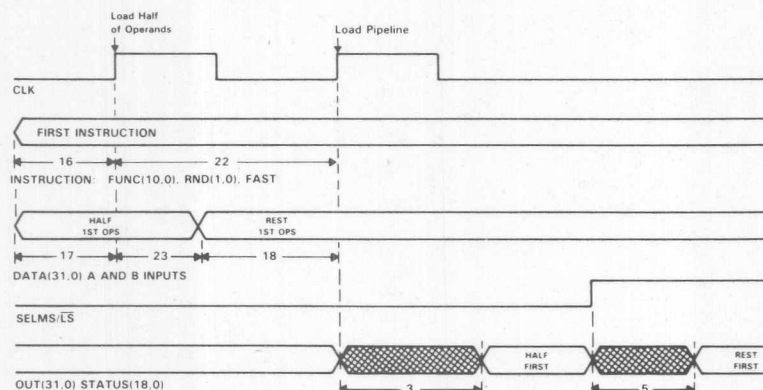
FIGURE 67. DOUBLE-PRECISION CHAINED MODE OPERATION, ALL REGISTERS DISABLED
(PIPES2-PIPES0 = 111, CLKMODE = 0)

PARAMETER MEASUREMENT INFORMATION



NOTE: Assume PIPES2-0 = 000, CONFIG1-0 = 01, ENRA = 1, ENRB = 1, SELMS/LS = X, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

FIGURE 66. SINGLE-PRECISION CHAINED MODE OPERATION, ALL REGISTERS ENABLED
(PIPES2-PIPES0 = 000, CLKMODE = X)



NOTE: Assume PIPES2-0 = 111, CONFIG1-0 = 11, ENRA = 1, ENRB = 1, OEY = 0, OEC = OES = 0, RESET = HALT = 1, TP1-0 = 11

FIGURE 67. DOUBLE-PRECISION CHAINED MODE OPERATION, ALL REGISTERS DISABLED
(PIPES2-PIPES0 = 111, CLKMODE = 0)

PARAMETER MEASUREMENT INFORMATION

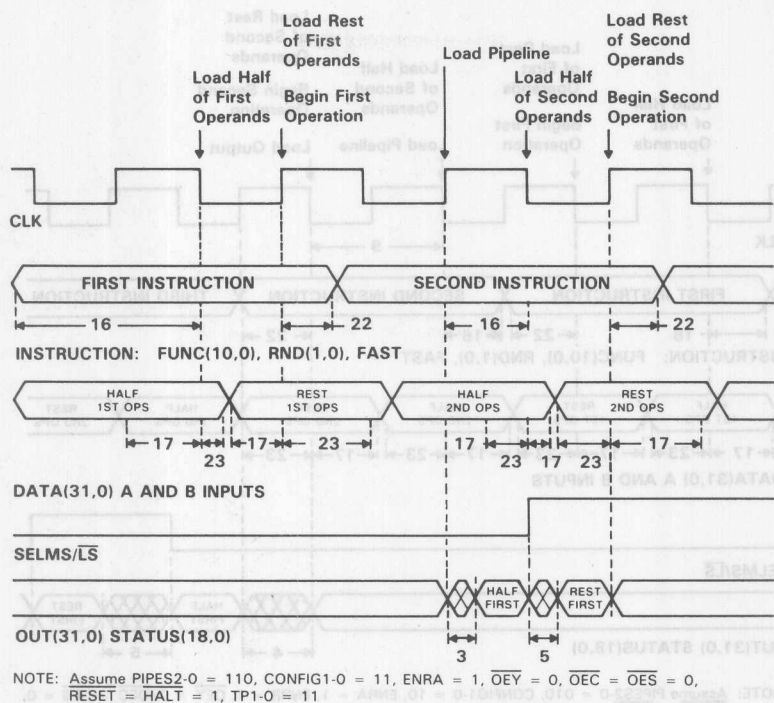


FIGURE 68. DOUBLE-PRECISION CHAINED MODE OPERATION, INPUT REGISTERS ENABLED
(PIPES2-PIPES0 = 110, CLKMODE = 1)

PARAMETER MEASUREMENT INFORMATION

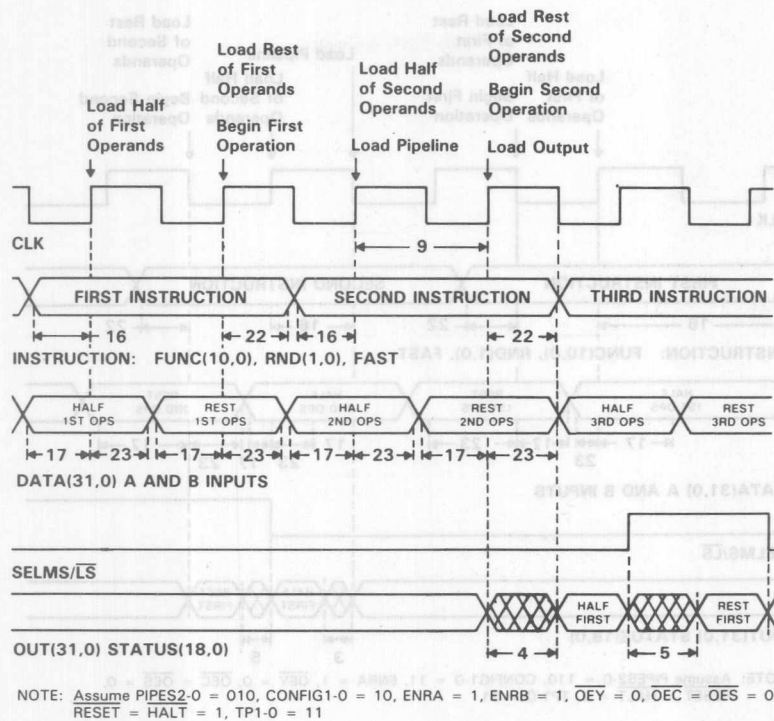


FIGURE 69. DOUBLE-PRECISION CHAINED MODE OPERATION, INPUT AND OUTPUT REGISTERS ENABLED (PIPES2-PIPES0 = 010, CLKMODE = 0)

PARAMETER MEASUREMENT INFORMATION

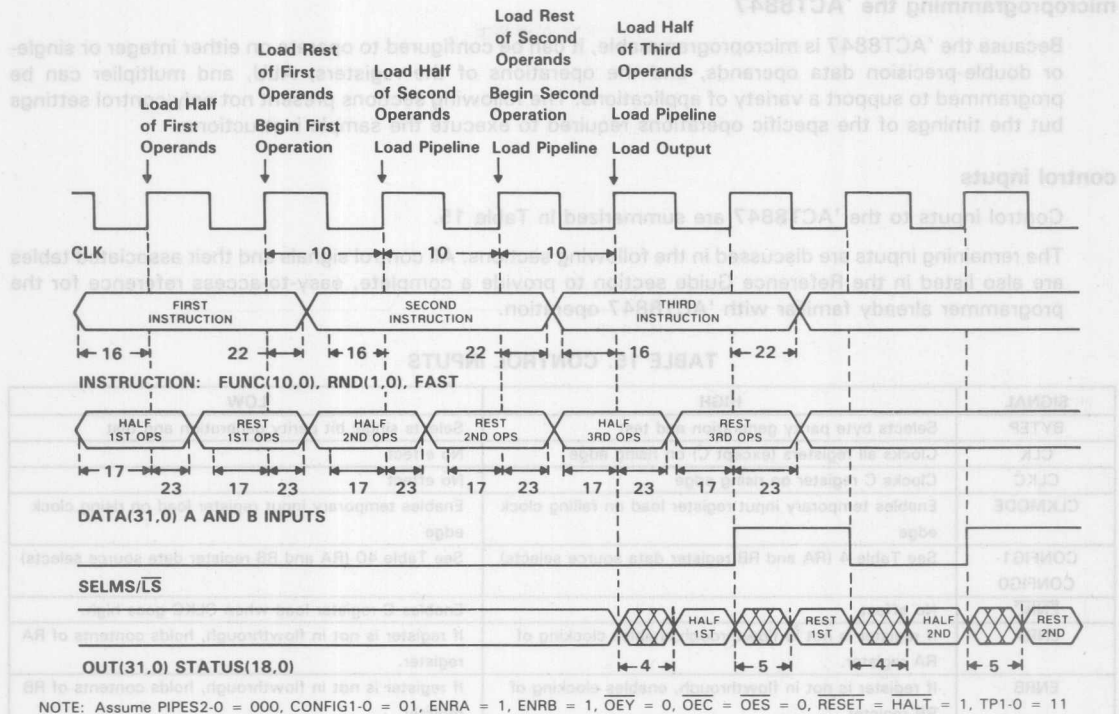


FIGURE 70. DOUBLE-PRECISION CHAINED MODE OPERATION, ALL REGISTERS ENABLED
(PIPES2-PIPES0 = 000, CLKMODE = 0)

| | | |
|---------------|---|--|
| TP1-TP0 | See Table 18 (Test Pin Control Inputs) | |
| SRCC | Selects multiplier result for input to C register. | Selects ALU result for input to C register. |
| SELST1 | See Table 14 (Status Output Selection) | |
| SELST2 | Selects M2M of 64-bit result for output on the Y bus. | Selects L2M of 64-bit result for output on the Y bus. |
| SELPH0 | (Selection) | |
| SELPH1 | See Tables 8 and 9 (Multiplier/ALU Operand Selection) | |
| RND1-RND0 | See Table 18 (Rounding Mode Control) | |
| RESET | no effect | Clears internal status, internal pipeline registers, and exception disable register. Does not affect other data registers. |
| PIPES2-PIPES0 | See Table 3 (Pipeline Mode Control) | |
| OEY | Disables Y bus | Enables Y bus |
| OES | Disables status outputs | Enables status outputs |
| OEC | Disables compare pins | Enables compare pins |
| HALT | no effect | Stalls device operation but does not affect registers, internal status, or status. C register loading is not disabled. |

PROGRAMMING INFORMATION

microprogramming the 'ACT8847

Because the 'ACT8847 is microprogrammable, it can be configured to operate on either integer or single- or double-precision data operands, and the operations of the registers, ALU, and multiplier can be programmed to support a variety of applications. The following sections present not only control settings but the timings of the specific operations required to execute the sample instructions.

control inputs

Control inputs to the 'ACT8847 are summarized in Table 15.

The remaining inputs are discussed in the following sections. All control signals and their associated tables are also listed in the Reference Guide section to provide a complete, easy-to-access reference for the programmer already familiar with 'ACT8847 operation.

TABLE 15. CONTROL INPUTS

| SIGNAL | HIGH | LOW |
|------------------|---|--|
| BYTEP | Selects byte parity generation and test | Selects single bit parity generation and test |
| CLK | Clocks all registers (except C) on rising edge | No effect |
| CLKC | Clocks C register on rising edge | No effect |
| CLKMODE | Enables temporary input register load on falling clock edge | Enables temporary input register load on rising clock edge |
| CONFIG1-CONFIG0 | See Table 4 (RA and RB register data source selects) | See Table 40 (RA and RB register data source selects) |
| ENRC | No effect | Enables C register load when CLKC goes high. |
| ENRA | If register is not in flowthrough, enables clocking of RA register. | If register is not in flowthrough, holds contents of RA register. |
| ENRB | If register is not in flowthrough, enables clocking of RB register | If register is not in flowthrough, holds contents of RB register |
| FAST | Places device in FAST mode | Places device in IEEE mode |
| FLOWC | Causes output value to bypass C register and appear on C register output bus. | No effect |
| HALT | No effect | Stalls device operation but does not affect registers, internal states, or status. C register loading is not disabled |
| \overline{OEC} | Disables compare pins | Enables compare pins |
| \overline{OES} | Disables status outputs | Enables status outputs |
| \overline{OEY} | Disables Y bus | Enables Y bus |
| PIPES2-PIPES0 | See Table 3 (Pipeline Mode Control) | See Table 3 (Pipeline Mode Control) |
| RESET | No effect | Clears internal states, status, internal pipeline registers, and exception disable register. Does not affect other data registers. |
| RND1-RND0 | See Table 16 (Rounding Mode Control) | See Table 16 (Rounding Mode Control) |
| SELOP7-SELOP0 | See Tables 8 and 9 (Multiplier/ALU Operand Selection) | See Tables 8 and 9 (Multiplier/ALU Operand Selection) |
| SELSMS/LS | Selects MSH of 64-bit result for output on the Y bus (no effect on single-precision operands) | Selects LSH of 64-bit result for output on the Y bus (no effect on single-precision operands) |
| SELST1-SELST0 | See Table 14 (Status Output Selection) | See Table 14 (Status Output Selection) |
| SRCC | Selects multiplier result for input to C register | Selects ALU result for input to C register |
| TP1-TPO | See Table 18 (Test Pin Control Inputs) | See Table 18 (Test Pin Control Inputs) |

PROGRAMMING INFORMATION

rounding modes

The 'ACT8847 supports the four IEEE standard rounding modes: round to nearest, round towards zero (truncate), round towards infinity (round up), and round towards minus infinity (round down). The rounding function is selected by control pins RND1 and RND0, as shown in Table 16.

TABLE 16. ROUNDING MODES

| RND1- RND0 | ROUNDING MODE SELECTED |
|---------------|--|
| 0 0 | Round towards nearest |
| 0 1 | Round towards zero (truncate) |
| 1 0 | Round towards infinity (round up) |
| 1 1 | Round towards negative infinity (round down) |

Rounding mode should be selected to minimize procedural errors which may otherwise accumulate and affect the accuracy of results. Rounding to nearest introduces a procedural error not exceeding half of the least significant bit for each rounding operation. Since rounding to nearest may involve rounding either upward or downward in successive steps, rounding errors tend to cancel each other.

In contrast, directed rounding modes may introduce errors approaching one bit for each rounding operation. Since successive rounding operations in a procedure may all be similarly directed, each introducing up to a one-bit error, rounding errors may accumulate rapidly, especially in single-precision operations.

FAST and IEEE modes

The device can be programmed to operate in FAST mode by asserting the FAST pin. In the FAST mode, all denormalized inputs and outputs are forced to zero.

Placing a zero on the FAST pin causes the chip to operate in IEEE mode. In this mode, the ALU can operate on denormalized inputs and return denormals. If a denorm is input to the multiplier, the DENIN flag will be asserted and the result will be invalid. Denormal numbers must be wrapped before being input to the multiplier. If the multiplier result underflows, a wrapped number will be output.

handling of denormalized numbers (FAST)

The FAST input selects the mode for handling denormalized inputs and outputs. When the FAST input is set low, the ALU accepts denormalized inputs but the multiplier generates an exception when a denormal is input. When FAST is set high, the DENIN status exception is disabled and all denormalized numbers, both inputs and results, are forced to zero.

A denormalized input has the form of a floating-point number with a zero exponent, a nonzero mantissa, and a zero in the leftmost bit of the mantissa (hidden or implicit bit). A denormalized number results from decrementing the biased exponent field to zero before normalization is complete. Since a denormalized number cannot be input to the multiplier, it must first be converted to a wrapped number by the ALU. When the mantissa of the denormal is normalized by shifting it left, the exponent field decrements from all zeros (wraps past zero) to a negative two's complement number (except in the case of 0.1XXX...), where the exponent is not decremented.

Exponent underflow is possible during multiplication of small operands even when the operands are not wrapped numbers. Setting FAST = 0 selects gradual underflow so that denormal inputs can be wrapped and wrapped results are not automatically discarded. When FAST is set high, denormal inputs and wrapped results are forced to zero immediately.

PROGRAMMING INFORMATION

When the multiplier is in IEEE mode and produces a wrapped number as its result, the result may be passed to the ALU and unwrapped. If the wrapped number can be unwrapped to an exact denormal, it can be output without causing the underflow status flag (UNDER) to be set. UNDER goes high when a result is an inexact denormal, and a zero is output from the FPU if the wrapped result is too small to represent as a denormal (smaller than the minimum denorm). Table 17 describes the handling of wrapped multiplier results and the status flags that are set when wrapped numbers are output from the multiplier.

TABLE 17. HANDLING WRAPPED MULTIPLIER OUTPUTS

| TYPE OF RESULT | STATUS FLAGS SET | | | NOTES |
|---------------------------------------|------------------|------|-------|--|
| | DENORM | INEX | RNDCO | |
| Wrapped, exact | 1 | 0 | 0 | Unwrap with 'Wrapped exact' ALU instruction |
| Wrapped, inexact | 1 | 1 | 0 | Unwrap with 'Wrapped inexact' ALU instruction |
| Wrapped, increased in magnitude | 1 | 1 | 1 | Unwrap with 'Wrapped rounded' ALU instruction |

When operating in chained mode, the multiplier may output a wrapped result to the ALU during the same clock cycle that the multiplier status is output. In such a case, the ALU cannot unwrap the operand prior to using it, for example, when accumulating the results of previous multiplications. To avoid this situation, the FPU can be operated in FAST mode to simplify exception handling during chained operations. Otherwise, wrapped outputs from the multiplier may adversely affect the accuracy of the chained operation, because a wrapped number may appear to be a large normalized number instead of a very small denormalized number.

Because of the latency associated with interpreting the FPU status outputs and determining how to process the wrapped output, it is necessary that a wrapped operand be stored external to the FPU (for example, in an external register file) and reloaded to the A port of the ALU for unwrapping and further processing.

stalling the device

Operation of the 'ACT8847 can be stalled nondestructively by means of the $\overline{\text{HALT}}$ signal. Bringing the $\overline{\text{HALT}}$ input low causes the device to inhibit the next rising clock edge. Register contents are unaltered when the device is stalled and normal operation resumes at the next low clock period after the $\overline{\text{HALT}}$ signal is set high.

Stalling the device does not stall the C register. If $\overline{\text{ENRC}}$ is low, CLKC will clock in data from the source selected by SRCC.

For some operations, such as a double-precision multiply with $\text{CLKMODE} = 1$, setting the $\overline{\text{HALT}}$ input low may interrupt loading of the RA, RB, and instruction registers, as well as stalling operation. In clock mode 1, the temporary register loads on the falling edge of the clock, but the $\overline{\text{HALT}}$ signal going low would prevent the RA, RB, and instruction registers from loading on the next rising clock edge. It is therefore necessary to have the instruction and data inputs on the pins when the $\overline{\text{HALT}}$ signal is set high again and normal operation resumes.

RESET

The RESET input is an active-low signal that asynchronously clears the internal states, status, and exception disable mask. Internal pipeline registers are cleared, but the RA, RB, and C registers are not. Operation resumes when RESET goes high again.

PROGRAMMING INFORMATION

test pins

Two pins, TP1-TP0, support system testing. These may be used, for example, to place all outputs in a high-impedance state, isolating the chip from the rest of the system (see Table 18).

TABLE 18. TEST PIN CONTROL INPUTS

| TP1-TP0 | OPERATION |
|---------|--|
| 0 0 | All outputs and I/Os are forced low |
| 0 1 | All outputs and I/Os are forced high |
| 1 0 | All outputs are placed in a high impedance state |
| 1 1 | Normal operation |

independent ALU operations

Configuration and operation of the 'ACT8847 can be selected to perform single- or double-precision floating-point and integer calculations in operating modes ranging from flowthrough to fully pipelined. Timing and sequences of operations are affected by settings of clock mode, data and status registers, input data configurations, and rounding mode, as well as the instruction inputs controlling the ALU and the multiplier.

Three modes of operation can be selected with inputs I10-I0, including independent ALU operation, independent multiplier operation, or simultaneous (chained) operation of ALU and multiplier. Each of these operating modes is treated separately in the following sections.

The ALU executes single- and double-precision operations which can be divided according to the number of operands involved, one or two. Tables 19 and 20 show independent ALU operations with one operand, along with the inputs I10-I0 which select each operation. Conversions from one format to another are handled in this mode, with the exception of adjustments to precision during two-operand ALU operations. The wrapping and unwrapping of operands is also done in this mode.

Most format conversions involve double-precision timing. Conversions between single- and double-precision floating-point format are treated as mixed-precision operations requiring two cycles to load the operands. A single-precision number is loaded in the upper half (MSH) of its input register. During integer to floating-point conversions, the integer input should be loaded into the upper half of the RA register. If converting from integer to double precision, then two cycles are required.

Logical shifts can be performed on integer operands using the instructions shown in Table 20. The data operand to be shifted is input from any valid operand source and the number of bit positions the operand is to be shifted is input only from the DB bus. The shift number on the DB bus should be in positive 32-bit integer format, although only the lowest eight bits are used. The shift number cannot be selected from sources other than the RB register and the shift number must be loaded on the same cycle as the instruction.

| RESULT | IN-TO | IS | IS | IS | IS | IS | IS |
|-----------------------------------|-------|------------|---------|------------|----|-------------|---------|
| Pass A operand | 0000 | 1 - Single | 0 - ALU | 0 - 32 bit | 0 | 1 - Integer | 0 - Not |
| Pass (A - A) operand | 0001 | Operands | result | complement | 0 | Integer | Chained |
| Negate A operand (2's complement) | 0010 | | | 1 - 32 | 0 | | |
| Pass B operand | 0011 | | | unwrapped | | | |
| Shift A operand left logical | 0100 | | | integer | | | |
| Shift A operand right logical | 0101 | | | | | | |
| Shift A operand right arithmetic | 0110 | | | | | | |

B operand is number of bit positions A is to be shifted. See instruction description for "Independent ALU Operations". The B operand must be input on the same cycle that shift is to be performed.

PROGRAMMING INFORMATION

TABLE 19. INDEPENDENT ALU OPERATIONS, SINGLE FLOATING-POINT OPERAND
(I10 = 0, I9 = 0, I6 = 0, I5 = 1)

| CHAINED OPERATION I10 | OPERAND FORMAT I9 | PRECISION RA I8 | PRECISION RB I7 | OUTPUT SOURCE I6 | OPERAND TYPE I5 | ABSOLUTE VALUE A I4 | ALU OPERATION | |
|-----------------------------|---------------------------|------------------------|--|------------------------|-----------------------|---------------------------|---------------|--|
| | | | | | | | I3-I0 | RESULT |
| 0 = Not Chained | 0 = Floating- point | 0 = A(SP) 1 = A(DP) | 0 = B(SP) 1 = B(DP) must equal I8 | 0 = ALU result | 1 = Single Operand | 0 = A 1 = A | 0000 | Pass A operand |
| | | | | | | | 0001 | Pass -A operand |
| | | | | | | | 0010 | 2's complement integer to floating-point conversion [†] |
| | | | | | | | 0011 | Floating-point to 2's complement integer conversion [†] |
| | | | | | | | 0100 | Move A operand (pass without NaN detect or exception flags active) |
| | | | | | | | 0101 | Pass B operand |
| | | | | | | | 0110 | Floating-point to floating- point conversion [†] |
| | | | | | | | 0111 | Floating-point to unsigned integer conversion [†] |
| | | | | | | | 1000 | Wrap (denormal) input operand |
| | | | | | | | 1010 | Unsigned integer to floating-point conversion [†] |
| | | | | | | | 1100 | Unwrap exact number |
| | | | | | | | 1101 | Unwrap inexact number |
| | | | | | | | 1110 | Unwrap rounded input |

[†]The precision of the integer to floating-point conversion is set by I8. If I8 = 1, the operation is timed like a double-precision operation, requiring clock edges to load.

[‡]This converts single-precision floating-point to double-precision floating-point and vice versa. If the I8 pin is low to indicate a single-precision input, the result of the conversion will be double precision. If the I8 pin is high, indicating a double-precision input, the result of the conversion will be single precision. This operation is timed like a double-precision operation, requiring 2 clock edges to load.

TABLE 20. INDEPENDENT ALU OPERATIONS, SINGLE INTEGER OPERAND
(I10 = 0, I9 = 1, I6 = 0, I5 = 1)

| CHAINED OPERATION I10 | OPERAND FORMAT/PRECISION | | | OUTPUT SOURCE I6 | OPERAND TYPE I5 | ALU OPERATION | |
|-----------------------------|--------------------------|----|-------------------------------|------------------------|------------------------|---------------|---|
| | I9 | I8 | I7 | | | I4-10 | RESULT |
| 0 = Not Chained | 1 = Integer | 0 | 0 = SP 2's complement | 0 = ALU result | 1 = Single Operands | 00000 | Pass A operand |
| | | 0 | 1 = SP unsigned integer | | | 00001 | Pass (– A) operand |
| | | | | | | 00010 | Negate A operand (1's complement) |
| | | | | | | 00101 | Pass B operand |
| | | | | | | 01000 | Shift A operand left logical [§] |
| | | | | | | 01001 | Shift A operand right logical [§] |
| | | | | | | 01101 | Shift A operand right arithmetic [§] |

[§]B operand is number of bit positions A is to be shifted (See instruction description for "Independent ALU Operations".) The B operand must be input on the same cycle that shift is to be performed.

PROGRAMMING INFORMATION

Tables 21 and 22 present independent ALU operations with two operands. When the operands are different in precision, one single and the other double, the settings of the precision selects I8-I7 will identify the single-precision operand so that it can automatically be reformatted to double-precision before the selected operation is executed, and the result of the operation will be double precision.

Precision of each data operand is indicated by the setting of instruction input I8 for single-operand ALU instructions, or the settings of I8-I7 for two-operand instructions. For single-operand instructions, I7 must be set equal to I8. When the ALU receives mixed-precision operands (one operand in single precision and the other in double precision), the single-precision data input is converted to double and the operation is executed in double precision. It is unnecessary to use the 'convert float-to-float' instruction to convert the single-precision operand prior to performing the desired operation on the mixed-precision operands. Setting I8 and I7 properly achieves the same effect without wasting an instruction cycle.

Timing for operations with mixed-precision operands is the same as for a corresponding double-precision operation. In a mixed-precision operation, the single-precision operand must be loaded into the upper half of its input register. If both operands are single precision, a single-precision result is output by the ALU. Operations on mixed-precision data inputs produce double-precision results. Mixed-precision operations do not apply to denormalized numbers.

TABLE 21. INDEPENDENT ALU OPERATIONS, TWO FLOATING-POINT OPERANDS
(I10 = 0, I9 = 0, I5 = 0)

| CHAINED OPERATION I10 | OPERAND FORMAT I9 | PRECISION RA I8 | PRECISION RB I7 | OUTPUT SOURCE I6 | OPERAND TYPE I5 | ABSOLUTE VALUE A I4 | ABSOLUTE VALUE B I3 | ABSOLUTE VALUE Y I2 | ALU OPERATION | |
|-----------------------------|---------------------------|------------------------|------------------------|------------------------|-----------------------|---------------------------|---------------------------|---------------------------|---------------|--------------|
| | | | | | | | | | I1-I0 | RESULT |
| 0 = Not Chained | 0 = Floating- point | 0 = A(SP) 1 = A(DP) | 0 = B(SP) 1 = B(DP) | 0 = ALU result | 0 = Two Operands | 0 = A 1 = A | 0 = B 1 = B | 0 = Y 1 = Y | 00 | A + B |
| | | | | | | | | | 01 | A - B |
| | | | | | | | | | 10 | Compare A, B |
| | | | | | | | | | 11 | B - A |

TABLE 22. INDEPENDENT ALU OPERATIONS, TWO INTEGER OPERANDS
(I10 = 0, I9 = 1, I6 = 0, I5 = 0)

| CHAINED OPERATION I10 | OPERAND FORMAT/PRECISION | | | OUTPUT SOURCE I6 | OPERAND TYPE I5 | ALU OPERATION | |
|-----------------------------|--------------------------|----|---|------------------------|-----------------------|---------------|----------------------------|
| | I9 | I8 | I7 | | | I4-I0 | RESULT |
| 0 = Not Chained | 1 = Integer | 0 | 0 = SP 2's complement 1 = SP unsigned integer | 0 = ALU result | 0 = Two Operands | 00000 | A + B |
| | | | | | | 00001 | A - B |
| | | | | | | 00010 | Compare A, B |
| | | | | | | 00011 | B - A |
| | | | | | | 01000 | Logical AND (A, B) |
| | | | | | | 01001 | Logical AND (A, NOT B) |
| | | | | | | 01010 | Logical AND (NOT A, B) |
| | | | | | | 01011 | Logical AND (NOT A, NOT B) |
| | | | | | | 01100 | Logical OR (A, B) |
| | | | | | | 01101 | Logical XOR (A, B) |
| | | | | | | | |
| | | | | | | | |

Two additional independent ALU operations may also be coded. The first of these is for loading the exception detect mask register.

PROGRAMMING INFORMATION

The exception detect mask register can be loaded with a mask to enable or disable selected status exceptions. Status bits for enabled exceptions are logically ORed, and when the result is true, the ED pin goes high. During chained operations, both multiplier and ALU results are ORed. During independent operation, the nonselected status results are forced to zero.

If the FPU is reset ($\overline{\text{RESET}} = 0$), the exception detect mask register is cleared. Table 23 describes the settings for the mask register load instruction and the status exceptions which can be enabled or disabled with the mask.

TABLE 23. LOADING THE EXCEPTION DISABLE MASK REGISTER

| INSTRUCTION INPUTS | RESULTS |
|-----------------------|---|
| I10-I7 = 0111 | Exception mask load instruction |
| I6 | 0 = Load ALU exception disable register 1 = Load multiplier exception disable register |
| I5† | 0 = IVAL exception enabled 1 = IVAL exception disabled |
| I4 | 0 = OVER exception enabled 1 = OVER exception disabled |
| I3 | 0 = UNDER exception enabled 1 = UNDER exception disabled |
| I2 | 0 = INEX exception enabled 1 = INEX exception disabled |
| I1 | 0 = DIVBY0 exception enabled 1 = DIVBY0 exception disabled† |
| I0 | 0 = DENORM exception enabled 1 = DENORM exception disabled |

†Disabling IVAL in multiplier exception mask register also disables DENIN exception

†Only significant when I6 = 1

The second additional independent ALU operation is the NOP (no operation). The table below shows the coding for the NOP instruction. This is treated as an integer operation and cannot be used to preload single- or double-precision floating-point operands.

TABLE 24. NOP INSTRUCTION

| I10-I0 | Operation |
|-------------|-----------|
| 01100000000 | NOP |

Because NOP, in effect, just prevents loading of the P or S registers, these registers must be enabled (PIPES2=0) for the NOP to work correctly.

Timing of a NOP instruction is the same as any single-precision ALU operation, taking one clock cycle per pipeline stage that is enabled. For example, when the ACT8847 is fully pipelined (PIPES2-PIPES0=000), a NOP's effect (preventing the overwriting of the P and S registers) will be seen on the third cycle. To hold the results of an operation on the Y bus for an extra cycle, the NOP instruction is inserted directly after the instruction whose results are to be held.

The NOP freezes the output register's contents until new results are to be loaded into these registers.

PROGRAMMING INFORMATION

independent multiplier operations

In this mode, the multiplier operates on two of five input sources which can be either single precision, double precision, or mixed. Multiplication, division, and square root may be coded as independent multiplier operations.

Operand precision is selected by I8 and I7, as for ALU operations. The multiplier can multiply the A and B operands, either operand with the absolute value of the other, or the absolute values of both operands. The result can also be negated when it is output. Operations involving absolute value or negated results are valid only when floating-point format is selected. If both operands are single precision, a single-precision result is output. Operations on mixed-precision data inputs produce double-precision results.

Floating-point operands may be normalized or wrapped numbers, as indicated by the settings for instruction inputs I1-I0. As shown in Table 25, the multiplier can be set to operate on the absolute value of either or both floating-point operands and the result of any operation can be negated when it is output from the multiplier. Converting a single-precision denormal number to double precision does not normalize or wrap the denormal, so it is still an invalid input to the multiplier. Independent multiplier operations are summarized in Tables 25 thru 27.

TABLE 25. INDEPENDENT MULTIPLIER OPERATIONS (I10 = 0, I6 = 1)

| CHAINED OPERATION I10 | OPERAND FORMAT/PRECISION | | | OUTPUT SOURCE I6 | MULTIPLY/ DIVIDE I5 | ABSOLUTE VALUE A I4† | ABSOLUTE VALUE B/ DIV/SQRT I3† | NEGATE RESULT I2† | WRAP A I1† | WRAP B I0† |
|-----------------------------|---------------------------------------|------------------------|---|-------------------------------|------------------------------|----------------------------|---|-------------------------|---|---|
| 0 = Not Chained | 0 = floating- point 1 = integer | 0 = A(SP) 1 = A(DP) | 0 = B(SP) 1 = B(DP) 0 = SP 2's complement 1 = SP unsigned integer | 1 = Multi- plier result | 0 = multiply 1 = Div/SQRT | 0 = A 1 = A | 0 = B 1 = B 0 = Div 1 = SQRT | 0 = Y 1 = -Y | 0 = Normal format 1 = A is a wrapped number | 0 = Normal format 1 = B is a wrapped number |

† See also Tables 11 and 12. Operations involving absolute values, negated results, or wrapped numbers are valid only when floating-point format is selected (I9 = 0).

‡ For square root operations, I7 must be equal to I8.

TABLE 26. INDEPENDENT MULTIPLY OPERATIONS
SELECTED BY I4-I2 (I10 = 0, I6 = 1, I5 = 0)

| ABSOLUTE VALUE A I4 | ABSOLUTE VALUE B I3 | NEGATE RESULT I2 | OPERATION SELECTED | |
|---------------------------|---------------------------|------------------------|--------------------|--------------|
| | | | I4-I2 | RESULTS† |
| 0 = A 1 = A | 0 = B 1 = B | 0 = Y 1 = -Y | 000 | A * B |
| | | | 001 | -(A * B) |
| | | | 010 | A * B |
| | | | 011 | -(A * B) |
| | | | 100 | A * B |
| | | | 101 | -(A * B) |
| | | | 110 | A * B |
| | | | 111 | -(A * B) |

† Operations involving absolute values or negated results are valid only when floating-point format is selected (I9 = 0).



PROGRAMMING INFORMATION

TABLE 27. INDEPENDENT DIVIDE/SQUARE ROOT OPERATIONS
SELECTED BY I4-I2 (I10 = 0, I6 = 1, I5 = 1)

| ABSOLUTE VALUE A I4 | ABSOLUTE VALUE B I3 | NEGATE RESULT I2 | OPERATION SELECTED | |
|---------------------------|---------------------------|------------------------|--------------------|-------------|
| | | | I4-I2 | RESULTS† |
| 0 = A | 0 = Divide | 0 = Y | 000 | A / B |
| 1 = A | 1 = SQRT | 1 = -Y | 001 | -(A / B) |
| | | | 010 | SQRT A |
| | | | 011 | -(SQRT A) |
| | | | 100 | A / B |
| | | | 101 | -(A / B) |
| | | | 110 | SQRT A |
| | | | 111 | -(SQRT A) |

†Operations involving absolute values or negated results are valid only when floating-point format is selected (I9 = 0).

chained multiplier/ALU operations

In chained mode, the 'ACT8847 performs simultaneous operations in the multiplier and the ALU. Operations not only include addition, subtraction, and multiplication, but also several optional operations which increase the flexibility of the device (see Table 28). Division and square root operations are not available in chained mode. Format conversions, absolute values, and wrapping or unwrapping of denormal numbers are also not available.

The B operand to the ALU can be set to zero so that the ALU passes the A operand unaltered. The B operand to the multiplier can be forced to the value 1 so that the A operand to the multiplier is passed unaltered.

Since in chained mode there are four operands but only two bits (I8 and I7) to select the operand precision, care must be taken with mixed-precision operations. The A input to the ALU and to the multiplier must be of the same precision; just as the B input to the ALU and to the multiplier must be of the same precision.

TABLE 28. CHAINED MULTIPLIER/ALU OPERATIONS (I10 = 1)

| CHAINED OPERATION I10 | OPERAND FORMAT/PRECISION | | | OUTPUT SOURCE I6 | ADD ZERO I5 | MULTIPLY BY ONE I4 | NEGATE ALU RESULT I3† | NEGATE MULTIPLIER RESULT I2† | ALU OPERATIONS | |
|-----------------------------|--------------------------|-----------|------------|------------------------|-------------------|--------------------------|--------------------------------|---------------------------------------|-------------------|--------|
| | I9 | I8 | I7 | | | | | | I1-I0 | RESULT |
| 1 = Chained | 0 = | 0 = A(SP) | 0 = B(SP) | 0 = | 0 = | 0 = | 0 = | 0 = | 00 | A + B |
| | floating- | 1 = A(DP) | 1 = B(DP) | ALU | Normal | Normal | Normal | Normal | 01 | A - B |
| | point | | | result | operation | operation | operation | operation | 10 | 2 - A |
| | | | | | 1 = | 1 = | 1 = | 1 = | 11 | B - A |
| | 1 = | 0 | 0 = SP 2's | 1 = | Forces | Forces | Negate | Negate | | |
| | integer | | complement | Multi- | B2 input | B1 input | ALU | multiplier | | |
| | | 0 | 1 = SP | plier | of ALU | of multi- | result | result | | |
| | | | unsigned | result | to zero | plier to | | | | |
| | | | integer | | | one | | | | |

†Operations involving negated results are valid only when floating-point format is selected (I9 = 0).

sample independent ALU microinstructions

The following independent ALU timing diagram examples show four register settings, ranging from fully flowthrough to fully pipelined. X = don't care.

PROGRAMMING INFORMATION

instruction timing

Table 29 details the number of clock cycles required to complete an operation in different pipelined modes. For more detail, see the sample microinstructions shown in the previous section.

Clock duration and output delay depend on the pipeline mode selected. See the note in the table and timing parameters listed at the beginning of this document.

TABLE 29. NUMBER OF CLOCKS REQUIRED TO COMPLETE AN OPERATION

| OPERATION | PIPES2-0 = 000 (t_{pd4}) | PIPES2-0 = 100 (t_{pd3}) | PIPES2-0 = 110 (t_{pd2}) | PIPES2-0 = 010 (t_{pd4}) |
|--|---------------------------------|---------------------------------|---------------------------------|---------------------------------|
| Single-Precision Floating-Point | | | | |
| ALU Operation or Multiply [†] | 3 | 2 | 1 | 2 |
| Divide | 8 | 7 | 7 | 8 |
| Square Root | 11 | 10 | 10 | 11 |
| Double-Precision Floating-Point | | | | |
| ALU Operation [†] | 4 | 3 | 2 | 3 |
| Multiply [‡] | 5 | 4 | 3 | 4 |
| Divide | 14 | 13 | 13 | 14 |
| Square Root | 17 | 16 | 16 | 17 |
| Integer | | | | |
| ALU Operation or Multiply [†] | 3 | 2 | 1 | 2 |
| Divide | 16 | 15 | 15 | 16 |
| Square Root | 20 | 19 | 19 | 20 |

Y output and status valid following this t_{pd} delay after the designated number of clocks

[†]Includes every conversion involving double-precision (DP \leftrightarrow SP or DP \leftrightarrow Integer)

[‡]Includes all chained mode operations

X = invalid

When using fast cycle times and double-precision operations, two cycles may be required to output and capture both halves of a double-precision result. To insure the result remains valid for two cycles, a NOP instruction may need to be inserted between the operations. Table 30 shows the number of NOPs necessary to insert into the instruction stream for fully pipelined operation (PIPES2-PIPES0 = 000).

PROGRAMMING INFORMATION

TABLE 30. NOPs INSERTED TO GUARANTEE THAT DOUBLE-PRECISION RESULTS REMAIN VALID FOR TWO CLOCK CYCLES (PIPES2-PIPE\$0 = 000)

| 1ST OPERATION | FOLLOWED BY 2ND OPERATION | # NOPs INSERTED BETWEEN OPERATIONS | # CYCLES RESULT IS VALID |
|---------------|------------------------------|---------------------------------------|-----------------------------|
| DP → 32 BIT | DP → 32 BIT | 0 | 2 |
| | 32 BIT → DP | 0 | 2 |
| | 32 BIT OP | 0 | 1 |
| | DP ALU | 0 | 2 |
| | DP Multiply | 0 | 2 |
| | DP Sqrt | 0 | 2 |
| | DP Divide | 0 | 2 |
| 32 BIT → DP | DP → 32 BIT | 0 | 2 |
| | 32 BIT → DP | 0 | 2 |
| | 32 BIT OP | 1 | 2 |
| | DP ALU | 0 | 2 |
| | DP Multiply | 0 | 2 |
| | DP Sqrt | 0 | 2 |
| | DP Divide | 0 | 2 |
| 32 BIT OP | DP → 32 BIT | 0 | 2 |
| | 32 BIT → DP | 0 | 2 |
| | 32 BIT OP | 0 | 1 |
| | DP ALU | 0 | 2 |
| | DP Multiply | 0 | 2 |
| | DP Sqrt | 0 | 2 |
| | DP Divide | 0 | 2 |
| DP ALU | DP → 32 BIT | 0 | 2 |
| | 32 BIT → DP | 0 | 2 |
| | 32 BIT OP | 1 | 2 |
| | DP ALU | 0 | 2 |
| | DP Multiply | 0 | 2 |
| | DP Sqrt | 0 | 2 |
| | DP Divide | 0 | 2 |
| DP Multiply | DP → 32 BIT | 1 | 2 |
| | 32 BIT → DP | 1 | 2 |
| | 32 BIT OP | 2† | 2 |
| | DP ALU | 1 | 2 |
| | DP Multiply | 0 | 2 |
| | DP Sqrt | 1 | 2 |
| | DP Divide | 1 | 2 |

NOTE: 32-bit operation refers to a single-precision floating-point or integer ALU operation or multiply, except conversion to or from double-precision. This assumes the instruction following a double-precision divide may begin loading on the 12th clock cycle, following a double-precision square root on the 15th cycle.

†The device will not load a single-precision operation on the first clock edge following this operation, so any single-precision instruction may be used. A NOP is recommended. The second instruction must be a NOP.

PROGRAMMING INFORMATION

TABLE 30. NOPs INSERTED TO GUARANTEE THAT DOUBLE-PRECISION RESULTS REMAIN VALID FOR TWO CLOCK CYCLES
(PIPES2-PIPE0 = 000) (Concluded)

| 1ST OPERATION | FOLLOWED BY 2ND OPERATION | # NOPs INSERTED BETWEEN OPERATIONS | # CYCLES RESULT IS VALID |
|---------------|------------------------------|---------------------------------------|-----------------------------|
| DP SQRT | DP → 32 BIT | 1 | 2 |
| | 32 BIT → DP | 1 | 2 |
| | 32 BIT OP | 2 [†] | 2 |
| | DP ALU | 1 | 2 |
| | DP Multiply | 0 | 2 |
| | DP Sqrt | 0 | 2 |
| | DP Divide | 0 | 2 |
| | DP Divide | 0 | 2 |
| DP Divide | DP → 32 BIT | 1 | 2 |
| | 32 BIT → DP | 1 | 2 |
| | 32 BIT OP | 2 [†] | 2 |
| | DP ALU | 1 | 2 |
| | DP Multiply | 0 | 2 |
| | DP Sqrt | 0 | 2 |
| | DP Divide | 0 | 2 |
| | DP Divide | 0 | 2 |

NOTE: 32-bit operation refers to a single-precision floating-point or integer ALU operation or multiply, except conversion to or from double-precision. This assumes the instruction following a double-precision divide may begin loading on the 12th clock cycle, following a double-precision square root on the 15th cycle.

[†]The device will not load a single-precision operation on the first clock edge following this operation, so any single-precision instruction may be used. A NOP is recommended. The second instruction must be a NOP.

exception and status handling

Exception and status flags for the 'ACT8847 were listed previously in Tables 12 and 13.

Output exception signals are provided to indicate both the source and type of the exception. DENORM, INEX, OVER, UNDER, and RNDCO indicate the exception type, and CHEX and SRCEX indicate the source of an exception. SRCEX indicates the source of a result as selected by instruction bit I6, and SRCEX is active whenever a result is output, not only when an exception is being signalled. The chained-mode exception signal CHEX indicates that an exception has been generated by the source not selected for output by I6. The exception type signalled by CHEX cannot be read unless status select controls SELST1-SELST0 are used to force status output from the deselected source.

Output exceptions may be due either to a result in an illegal format or to a procedural error. Results too large or too small to be represented in the selected precision are signalled by OVER and UNDER. When INF is high, the output is the IEEE representation of infinity. Any ALU output which has been increased in magnitude by rounding causes INEX to be set high. DENORM is set when the multiplier output is wrapped or the ALU output is denormalized. DENORM is also set high when an illegal operation on an integer is performed. Wrapped outputs from the multiplier may be inexact or increased in magnitude by rounding, which may cause the INEX and RNDCO status signals to be set high. A denormal output from the ALU (DENORM = 1) may also cause INEX to be set, in which case UNDER is also signalled.

Ordinarily, SELST1-SELST0 are set high so that status selection defaults to the output source selected by instruction input I6. The ALU is selected as the output source when I6 is low, and the multiplier when I6 is high.



PROGRAMMING INFORMATION

When the device operates in chained mode, it may be necessary to read the status results not associated with the output source. As shown in Table 14, SELST1-SELST0 can be used to read the status of either the ALU or the multiplier regardless of the I6 setting.

Status results are registered only when the output (P and S) registers are enabled (PIPES2 = 0). Otherwise, the status register is transparent. In either case, to read the status outputs, the output enables (\overline{OES} , \overline{OEC} , or both) must be low.

Status flags are provided to signal both floating-point and integer results. Integer status is provided using AEQB for zero, NEG for sign, and OVER for overflow/carryout.

Several status exceptions are generated by illegal data or instruction inputs to the FPU. Input exceptions may cause the following signals to be set high: IVAL, DIVBY0, DENIN, and STEX1-STEX0. If the IVAL flag is set, either an invalid operation such as the square root of $-|X|$, has been requested or a NaN (Not a Number) has been input. When DENIN is set, a denormalized number has been input to the multiplier. DIVBY0 is set when the divisor is zero. STEX1-STEX0 indicate which port (RA, RB, or both) is the source of the exception when either a denormal is input to the multiplier (DENIN = 1) or a NaN (IVAL = 1) is input to the multiplier or the ALU.

NaN inputs are all treated as IEEE signalling NaNs, causing the IVAL flag to be set. When output from the FPU, the fraction field from a NaN is set high (all 1s) and the sign bit is 0, regardless of the original fraction and sign fields of the input NaN.

When the 'ACT8847 outputs a NaN, it is always in the form of a signalling NaN along with the IVAL (Invalid) and appropriate STEX flag set high (except for the MOVE A instruction which passes any operand as is without setting exception flags).

Certain operations involving floating-point zeros and infinities are invalid, causing the 'ACT8847 to set the IVAL flag and output a NaN. Operations involving zero and infinity are detailed below.

When executing a floating-point to integer conversion instruction which returns an inexact 8000 0000, the overflow flag is set.

A floating-point zero is represented by an all zero exponent and fraction field. The sign bit may be 0 or 1, to represent +0 OR -0, respectively.

Zero divided by zero is an invalid operation. The result is a NaN with the IVAL and DIVBY0 flags set. Any other number divided by zero results in the appropriately signed infinity with the DIVBY0 flag set.

For operations with floating-point zeros: ± 0 multiplied by any number is the appropriately signed 0.

$+0 + (-0) = +0$
 $+0 + (+0) = +0$
 $-0 + (-0) = -0$
 $-0 + (+0) = +0$
 $+0 - (-0) = +0$
 $+0 - (+0) = +0$
 $-0 - (-0) = +0$
 $-0 - (+0) = -0$
 $\sqrt{-|0|} = -0$

Floating-point infinity is represented by an all 1 exponent field with an all 0 fraction field. The sign bit determines positive or negative infinity (0 or 1 respectively).

Infinity divided by infinity is an invalid operation, setting the IVAL flag and resulting in a NaN output. Division of infinity by any other number results in the appropriately signed infinity. Division of any number (except infinity or zero) by infinity results in an appropriately signed zero. Infinity divided by zero results in the appropriately signed infinity with the DIVBY0 flag set.

PROGRAMMING INFORMATION

For invalid operations with infinity listed below, the output is a signalling NaN with the IVAL flag set.

\pm infinity multiplied by ± 0
 \pm infinity divided by ± 0
 $+$ infinity + ($-$ infinity)
 $-$ infinity + ($+$ infinity)
 $+$ infinity - ($+$ infinity)
 $-$ infinity - ($-$ infinity)

Any other number added to or multiplied by infinity results in the appropriately signed infinity as output.

CUSTOMER SUPPORT

support tools

Texas Instruments has developed functional evaluation models of the 'ACT8847 in software which permit designers to simulate operation of the FPU. To evaluate the functions of an FPU, a designer can create a microprogram with sample data inputs, and the simulator will emulate FPU operation to produce sample data output files, as well as several diagnostic displays to show specific aspects of device operation. Sample microprogram sequences are included.

design support

Texas Instruments Regional Technology Centers, staffed with systems-oriented engineers, offer a training course to assist users of TI LSI products and their application to digital processor systems. Specific attention is given to the understanding and generation of design techniques which implement efficient algorithms designed to match high-performance hardware capabilities with desired performance levels.

Information on VLSI devices and product support can be obtained from the following Regional Technology Centers:

| | |
|---|---|
| Atlanta Texas Instruments Incorporated 3300 N.E. Expressway, Building 8 Atlanta, GA 30341 404/662-7945 | Chicago Texas Instruments Incorporated 515 Algonquin Arlington Heights, IL 60005 312/640-2909 |
| Boston Texas Instruments Incorporated 950 Winter Street, Suite 2800 Waltham, MA 02154 617/895-9100 | Dallas Texas Instruments Incorporated 10001 E. Campbell Road Richardson, TX 75081 214/680-5066 |
| Northern California Texas Instruments Incorporated 5353 Betsy Ross Drive Santa Clara, CA 95054 408/748-2220 | Southern California Texas Instruments Incorporated 17891 Cartwright Drive Irvine, CA 92714 714/660-8140 |

design expertise

Texas Instruments can provide in-depth technical design assistance through consultations with contract design services. Contact your local Field Sales Engineer for current information or contact Datapath VLSI Products Systems Engineering at 214/997-3970.



REFERENCE GUIDE

instruction inputs

Operations are summarized in Tables 31 thru 39.

TABLE 31. INDEPENDENT ALU OPERATIONS, SINGLE FLOATING-POINT OPERAND

| ALU OPERATION ON A OPERAND | INSTRUCTION INPUTS I10-I0 | NOTES |
|---|------------------------------|---|
| Pass A operand | 00x x01x 0000 | |
| Pass -A operand | 00x x01x 0001 | |
| Convert from 2's complement integer to floating-point [†] | 00x x010 0010 | |
| Convert from floating- point to 2's complement integer [†] | 00x x01x 0011 | x = Don't care 18 selects precision of A operand |
| Move A operand (pass without NaN detect or status flags active) | 00x x01x 0100 | 0 = A (SP) 1 = A (DP) |
| Pass B operand | 00x x01x 0101 | 17 selects precision of B operand and must equal 18. |
| Convert from floating- point to floating-point (adjusts precision of input: SP → DP, DP → SP) [‡] | 00x x01x 0110 | 14 selects absolute value of a operand: 0 = A 1 = A |
| Floating-point to unsigned integer conversion [†] | 00x x01x 0111 | During integer to floating- point conversion, A is not allowed as a result. |
| Wrap denormal operand | 00x x01x 1000 | |
| Unsigned integer to floating-point conversion [†] | 00x x01x 1010 | |
| Unwrap exact number | 00x x01x 1100 | |
| Unwrap inexact number | 00x x01x 1101 | |
| Unwrap rounded input | 00x x01x 1110 | |

[†]During this operation, 18 selects the precision of the result. If the conversion involves doubleprecision, the operation requires 2 cycles to load.

[‡]Requires 2 cycles to load the operation, even if input is SP.

REFERENCE GUIDE

TABLE 32. INDEPENDENT ALU OPERATIONS, TWO FLOATING-POINT OPERANDS

| ALU OPERATIONS AND OPERANDS | INSTRUCTION INPUTS I10-I0 | NOTES |
|--------------------------------|------------------------------|---|
| Add A + B | 00x x000 0x00 | |
| Add A + B | 00x x001 0x00 | x = Don't Care |
| Add A + B | 00x x000 1x00 | 18 selects precision of A operand: |
| Add A + B | 00x x001 1x00 | 0 = A (SP) 1 = A (DP) |
| Subtract A - B | 00x x000 0x01 | 17 selects precision of B operand: |
| Subtract A - B | 00x x001 0x01 | 0 = B (SP) 1 = B (DP) |
| Subtract A - B | 00x x000 1x01 | 12 selects either Y or its absolute value: |
| Subtract A - B | 00x x001 1x01 | 0 = Y 1 = Y |
| Compare A, B | 00x x000 0x10 | |
| Compare A , B | 00x x001 0x10 | |
| Compare A, B | 00x x000 1x10 | |
| Compare A , B | 00x x001 1x10 | |
| Subtract B - A | 00x x000 0x11 | |
| Subtract B - A | 00x x001 0x11 | |
| Subtract B - A | 00x x000 1x11 | |
| Subtract B - A | 00x x001 1x11 | |

TABLE 33. INDEPENDENT ALU OPERATIONS, ONE INTEGER OPERAND

| ALU OPERATION ON A OPERAND | INSTRUCTION INPUTS I10-I0 | NOTES |
|---|------------------------------|---|
| Pass A operand | 010 x010 0000 | x = Don't Care |
| Pass -A operand (2's complement) [‡] | 010 x010 0001 | 17 selects format of A or B integer operand: |
| Negate A operand (1's complement) | 010 x010 0010 | 0 = Single-precision 2's complement |
| Pass B operand | 010 x010 0101 | 1 = Single-precision unsigned integer |
| Shift left logical [†] | 010 x010 1000 | |
| Shift right logical [†] | 010 x010 1001 | |
| Shift right arithmetic [†] | 010 x010 1101 | |

[†]B operand is number of bit positions A is to be shifted and must be input on the same cycle as the instruction.

[‡]Pass (-A) of unsigned integer takes 1's complement.

REFERENCE GUIDE

TABLE 34. INDEPENDENT ALU OPERATIONS, TWO INTEGER OPERANDS

| ALU OPERATIONS AND OPERANDS | INSTRUCTION INPUTS I10-I0 | NOTES |
|--------------------------------|------------------------------|---|
| Add A + B | 010 x000 0000 | x = Don't Care I7 selects format of A and B operands: 0 = Single-precision 2's complement 1 = Single-precision unsigned integer |
| Subtract A - B | 010 x000 0001 | |
| Compare A, B | 010 x000 0010 | |
| Subtract B - A | 010 x000 0011 | |
| Logical AND A, B | 010 x000 1000 | |
| Logical AND A, NOT B | 010 x000 1001 | |
| Logical AND NOT A, B | 010 x000 1010 | |
| Logical OR A, B | 010 x000 1100 | |
| Logical XOR A, B | 010 x000 1101 | |

TABLE 35. INDEPENDENT FLOATING-POINT MULTIPLY OPERATIONS

| MULTIPLIER OPERATION AND OPERANDS | INSTRUCTION INPUTS I10-I0 | NOTES |
|--------------------------------------|------------------------------|--|
| Multiply A * B | 00x x100 00xx | x = Don't Care I8 selects A operand precision (0 = SP, 1 = DP) I7 selects B operand precision (0 = SP, 1 = DP) I1 selects A operand format (0 = Normal, 1 = Wrapped) I0 selects B operand format (0 = Normal, 1 = Wrapped) |
| Multiply -(A * B) | 00x x100 01xx | |
| Multiply A * B | 00x x100 10xx | |
| Multiply -(A * B) | 00x x100 11xx | |
| Multiply A * B | 00x x101 00xx | |
| Multiply -(A * B) | 00x x101 01xx | |
| Multiply A * B | 00x x101 10xx | |
| Multiply -(A * B) | 00x x101 11xx | |

TABLE 36. INDEPENDENT FLOATING-POINT DIVIDE/SQUARE ROOT OPERATIONS

| MULTIPLIER OPERATION AND OPERANDS† | INSTRUCTION INPUTS I10-I0 | NOTES |
|---------------------------------------|------------------------------|--|
| Divide A / B | 00x x110 0xxx | x = Don't Care I8 selects A operand precision and I7 selects B operand precision (0 = SP, 1 = DP) I2 negates multiplier result (0 = Normal, 1 = Negated) I1 selects A operand format and I0 selects B operand format (0 = Normal, 1 = Wrapped) |
| SQRT A | 00x x110 1xxx | |
| Divide A / B | 00x x111 0xxx | |
| SQRT A | 00x x111 1xxx | |

†I7 should be equal to I8 for square root operations

REFERENCE GUIDE

TABLE 37. INDEPENDENT INTEGER MULTIPLY/DIVIDE/SQUARE ROOT OPERATIONS

| MULTIPLIER OPERATION AND OPERANDS [†] | INSTRUCTION INPUTS I10-I0 | NOTES |
|---|------------------------------|--|
| Multiply A * B | 010 x100 0000 | x = Don't care |
| Divide A / B | 010 x110 0000 | I7 selects operand format: |
| SQRT A | 010 x110 1000 | 0 = SP 2's complement 1 = SP unsigned integer |

[†]Operations involving absolute values, wrapped operands, or negated results are valid only when floating-point format is selected (I9 = 0).

TABLE 38. CHAINED MULTIPLIER/ALU FLOATING-POINT OPERATIONS[‡]

| CHAINED OPERATIONS | | OUTPUT SOURCE | INSTRUCTION INPUTS I10-I0 | NOTES |
|--------------------|-------|------------------|------------------------------|-------------------------|
| MULTIPLIER | ALU | | | |
| A * B | A + B | ALU | 10x x000 xx00 | |
| A * B | A + B | Multiplier | 10x x100 xx00 | |
| A * B | A - B | ALU | 10x x000 xx01 | |
| A * B | A - B | Multiplier | 10x x100 xx01 | |
| A * B | 2 - A | ALU | 10x x000 xx10 | x = Don't Care |
| A * B | 2 - A | Multiplier | 10x x100 xx10 | I8 selects precision of |
| A * B | B - A | ALU | 10x x000 xx11 | RA inputs: |
| A * B | B - A | Multiplier | 10x x100 xx11 | 0 = RA (SP) |
| A * B | A + 0 | ALU | 10x x010 xx00 | 1 = RA (DP) |
| A * B | A + 0 | Multiplier | 10x x110 xx00 | I7 selects precision of |
| A * B | 0 - A | ALU | 10x x010 xx11 | RB inputs: |
| A * B | 0 - A | Multiplier | 10x x110 xx11 | 0 = RB (SP) |
| A * 1 | A + B | ALU | 10x x001 xx00 | 1 = RB (DP) |
| A * 1 | A + B | Multiplier | 10x x101 xx00 | I3 negates ALU |
| A * 1 | A - B | ALU | 10x x001 xx01 | result: |
| A * 1 | A - B | Multiplier | 10x x101 xx01 | 0 = Normal |
| A * 1 | 2 - A | ALU | 10x x001 xx10 | 1 = Negated |
| A * 1 | 2 - A | Multiplier | 10x x101 xx10 | I2 negates multiplier |
| A * 1 | B - A | ALU | 10x x001 xx11 | result: |
| A * 1 | B - A | Multiplier | 10x x101 xx11 | 0 = Normal |
| A * 1 | A + 0 | ALU | 10x x011 xx00 | 1 = Negated |
| A * 1 | A + 0 | Multiplier | 10x x111 xx00 | |
| A * 1 | 0 - A | ALU | 10x x011 xx11 | |
| A * 1 | 0 - A | Multiplier | 10x x111 xx11 | |

[‡]The I10-I0 setting 1xx xx1x xx10 is invalid, since it attempts to force the B operand of the ALU to both 0 and 2 simultaneously.

SN74ACT8847
64-BIT FLOATING-POINT UNIT

REFERENCE GUIDE

TABLE 39. CHAINED MULTIPLIER/ALU INTEGER OPERATIONS

| CHAINED OPERATIONS | | OUTPUT SOURCE | INSTRUCTION INPUTS I10-I0 | NOTES |
|--------------------|-------|------------------|------------------------------|---|
| MULTIPLIER | ALU | | | |
| A * B | A + B | ALU | 110 x000 0000 | |
| A * B | A + B | Multiplier | 110 x100 0000 | |
| A * B | A - B | ALU | 110 x000 0001 | |
| A * B | A - B | Multiplier | 110 x100 0001 | |
| A * B | 2 - A | ALU | 110 x000 0010 | |
| A * B | 2 - A | Multiplier | 110 x100 0010 | |
| A * B | B - A | ALU | 110 x000 0011 | |
| A * B | B - A | Multiplier | 110 x100 0011 | |
| A * B | A + 0 | ALU | 110 x010 0000 | x = Don't Care I7 selects format of A and B operands: 0 = SP 2's complement 1 = SP unsigned integer |
| A * B | A + 0 | Multiplier | 110 x110 0000 | |
| A * B | 0 - A | ALU | 110 x010 0011 | |
| A * B | 0 - A | Multiplier | 110 x110 0011 | |
| A * 1 | A + B | ALU | 110 x001 0000 | |
| A * 1 | A + B | Multiplier | 110 x101 0000 | |
| A * 1 | A - B | ALU | 110 x001 0001 | |
| A * 1 | A - B | Multiplier | 110 x101 0001 | |
| A * 1 | 2 - A | ALU | 110 x001 0010 | |
| A * 1 | 2 - A | Multiplier | 110 x101 0010 | |
| A * 1 | B - A | ALU | 110 x001 0011 | |
| A * 1 | B - A | Multiplier | 110 x101 0011 | |
| A * 1 | A + 0 | ALU | 110 x011 0000 | |
| A * 1 | A + 0 | Multiplier | 110 x111 0000 | |
| A * 1 | 0 - A | ALU | 110 x011 0011 | |
| A * 1 | 0 - A | Multiplier | 110 x111 xx11 | |

REFERENCE GUIDE

input configuration

CONFIG1-CONFIG0 control the order in which double-precision operands are loaded, as shown in Table 40.

TABLE 40. DOUBLE-PRECISION INPUT DATA CONFIGURATION MODES

| CONFIG1 | CONFIG0 | LOADING SEQUENCE | | | |
|---------|----------------|--|-----------------|--|-----------------|
| | | DATA LOADED INTO TEMP REGISTER ON FIRST CLOCK AND RA/RB REGISTERS ON SECOND CLOCK [†] | | DATA LOADED INTO RA/RB REGISTERS ON SECOND CLOCK | |
| | | DA | DB | DA | DB |
| 0 | 0 | B operand (MSH) | B operand (LSH) | A operand (MSH) | A operand (LSH) |
| 0 | 1 [‡] | A operand (LSH) | B operand (LSH) | A operand (MSH) | B operand (MSH) |
| 1 | 0 | A operand (MSH) | B operand (MSH) | A operand (LSH) | B operand (LSH) |
| 1 | 1 | A operand (MSH) | A operand (LSH) | B operand (MSH) | B operand (LSH) |

[†] On the first active clock edge (see CLKMODE), data in this column is loaded into the temporary register. On the next rising edge, operands in the temporary register and the DA/DB buses are loaded into the RA and RB registers.

[‡] Use CONFIG1-0 = 01 as normal single-precision input configuration.

operand source select

Multiplier and ALU operands are selected by SELOP7-SELOP0 as shown in Tables 41 and 42.

TABLE 41. MULTIPLIER INPUT SELECTION

| A1 (MUX1) INPUT | | | B1 (MUX2) INPUT | | |
|-----------------|--------|-----------------------------|-----------------|--------|-----------------------------|
| SELOP7 | SELOP6 | OPERAND SOURCE [†] | SELOP5 | SELOP4 | OPERAND SOURCE [†] |
| 0 | 0 | Reserved | 0 | 0 | Reserved |
| 0 | 1 | C register | 0 | 1 | C register |
| 1 | 0 | ALU feedback | 1 | 0 | Multiplier feedback |
| 1 | 1 | RA input register | 1 | 1 | RB input register |

[†] For division or square root operations, only RA and RB registers can be selected as sources.

TABLE 42. ALU INPUT SELECTION

| A2 (MUX3) INPUT | | | B2 (MUX4) INPUT | | |
|-----------------|--------|-----------------------------|-----------------|--------|-----------------------------|
| SELOP3 | SELOP2 | OPERAND SOURCE [†] | SELOP1 | SELOP0 | OPERAND SOURCE [†] |
| 0 | 0 | Reserved | 0 | 0 | Reserved |
| 0 | 1 | C register | 0 | 1 | C register |
| 1 | 0 | Multiplier feedback | 1 | 0 | ALU feedback |
| 1 | 1 | RA input register | 1 | 1 | RB input register |

[†] For division or square root operations, only RA and RB registers can be selected as sources.

REFERENCE GUIDE

pipeline control

Pipelining levels are turned on by PIPES2-PIPES0 as shown in Table 43.

TABLE 43. PIPELINE CONTROLS (PIPES2-PIPES0)

| PIPES2- PIPES0 | ROUNDING MODE SELECTED |
|-------------------|---|
| X X 0 | Enables input registers (RA, RB) |
| X X 1 | Makes input registers (RA, RB) transparent |
| X 0 X | Enables pipeline registers |
| X 1 X | Makes pipeline registers transparent |
| 0 X X | Enables output registers (PREG, SREG, Status) |
| 1 X X | Makes output registers (PREG, SREG, Status) transparent |

round control

RND1-RND0 select the rounding mode as shown in Table 44.

TABLE 44. ROUNDING MODES

| RND1- RND0 | ROUNDING MODE SELECTED |
|---------------|--|
| 0 0 | Round towards nearest |
| 0 1 | Round towards zero (truncate) |
| 1 0 | Round towards infinity (round up) |
| 1 1 | Round towards negative infinity (round down) |

status output selection

SELST1-SELST0 choose the status output as shown in Table 45.

TABLE 45. STATUS OUTPUT SELECTION (CHAINED MODE)

| SELST1- SELST0 | STATUS SELECTED |
|-------------------|---|
| 0 0 | Logical OR of ALU and multiplier exceptions (bit by bit) |
| 0 1 | Selects multiplier status |
| 1 0 | Selects ALU status |
| 1 1 | Normal operation (selection based on result source specified by I6 input) |

test pin control

Testing is controlled by TP1-TP0 as shown in Table 46.

TABLE 46. TEST PIN CONTROL INPUTS

| TP1- TP0 | OPERATION |
|-------------|--|
| 0 0 | All outputs and I/Os are forced low |
| 0 1 | All outputs and I/Os are forced high |
| 1 0 | All outputs are placed in a high impedance state |
| 1 1 | Normal operation |

REFERENCE GUIDE

miscellaneous control inputs

The remaining control inputs are shown in Table 47.

TABLE 47. MISCELLANEOUS CONTROL INPUTS

| SIGNAL | HIGH | LOW |
|-------------------------------|---|--|
| BYTEP | Selects byte parity generation and test | Selects single bit parity generation and test |
| CLKMODE | Enables temporary input register load on falling clock edge | Enables temporary input register load on rising clock edge |
| $\overline{\text{ENRC}}$ | No effect | Enables C register load when CLKC goes high |
| ENRA | If register is not in flowthrough, enables clocking of RB register. | If register is not in flowthrough, holds contents of RA register. |
| ENRB | If register is not in flowthrough, enables clocking of RB register. | If register is not in flowthrough, holds contents of RB register. |
| FAST | Places device in FAST mode | Places device in IEEE mode |
| FLOWC | Causes output value to bypass C register and appear on C register output bus. | No effect |
| $\overline{\text{HALT}}$ | No effect | Stalls device operation but does not affect registers, internal states, or status |
| $\overline{\text{OEC}}$ | Disables compare pins | Enables compare pins |
| $\overline{\text{OES}}$ | Disables status outputs | Enables status outputs |
| $\overline{\text{OEY}}$ | Disables Y bus | Enables Y bus |
| $\overline{\text{RESET}}$ | No effect | Clears internal states, status, internal pipeline registers, and exception disable register. Does not affect other data registers. |
| SELMS/ $\overline{\text{LS}}$ | Selects MSH of 64-bit result for output on the Y bus (no effect on single-precision operands) | Selects LSH of 64-bit result for output on the Y bus (no effect on single-precision operands) |
| SRCC | Selects multiplier result for input to C register | Selects ALU result for input to C register |

GLOSSARY

biased exponent — The true exponent of a floating-point number plus a constant called the exponent field's excess. In IEEE data format, the excess or bias is 127 for single-precision numbers and 1023 for double-precision numbers.

denormalized number (denorm) — A number with an exponent equal to zero and a nonzero fraction field, with the implicit leading (leftmost) bit of the fraction field being 0.

NaN (not a number) — Data that has no mathematical value. The 'ACT8847 produces a NaN whenever an invalid operation such as $0 * \infty$ is executed. The output format for an NaN is an exponent field of all ones, a fraction field of all ones, and a zero sign bit. Any number with an exponent of all ones and a nonzero fraction is treated as a NaN on the input.

normalized number — A number in which the exponent field is between 1 and 254 (single-precision) or 1 and 2046 (double-precision). The implicit leading bit is 1.

wrapped number — A number created by normalizing a denormalized number's fraction field and subtracting from the exponent the number of shift positions required to do so. The exponent is encoded as a two's complement negative number.

REFERENCE GUIDE

Miscellaneous control inputs

The remaining control inputs are shown in Table 47.

TABLE 47. MISCELLANEOUS CONTROL INPUTS

| SIGNAL | FUNCTION | LOW |
|---------|--|--|
| SRCC | Selects register output for input to C register | Selects register output for input to C register |
| SRMATH | Selects MATH of 84-bit result for output on the Y bus (no effect on single-precision operands) | Selects MATH of 84-bit result for output on the Y bus (no effect on single-precision operands) |
| RESET | No effect | No effect |
| DEY | Decreases Y bus | Decreases Y bus |
| DOE | Decreases status outputs | Decreases status outputs |
| DEEC | Decreases compare pins | Decreases compare pins |
| DATA | No effect | No effect |
| FLOWC | C register output bus | C register output bus |
| FAST | Places device in FAST mode | Places device in FAST mode |
| ENMS | If register is not in flowthrough, enables clocking of RB register | If register is not in flowthrough, enables clocking of RB register |
| ENRA | If register is not in flowthrough, enables clocking of RA register | If register is not in flowthrough, enables clocking of RA register |
| ENRC | No effect | No effect |
| CLKMODE | Enables temporary input register load on falling clock edge | Enables temporary input register load on falling clock edge |
| BYTYP | Selects byte generation and test | Selects single or double generation and test |

GLOSSARY

biased exponent — The true exponent of a floating-point number plus a constant called the exponent field's excess. In IEEE data format, the excess or bias is 127 for single-precision numbers and 1023 for double-precision numbers.

denormalized number (denorm) — A number with an exponent equal to zero and a nonzero fraction field, with the implicit leading leftmost bit of the fraction field being 0.

NaN (not a number) — Data that has no mathematical value. The ACT884Z produces a NaN whenever an invalid operation such as 0 ÷ 0 is executed. The output format for a NaN is an exponent field of all ones, a fraction field of all ones, and a zero sign bit. Any number with an exponent of all ones and a nonzero fraction is treated as a NaN on the input.

normalized number — A number in which the exponent field is between 1 and 254 (single-precision) or 1 and 2046 (double-precision). The implicit leading bit is 1.

unwrapped number — A number created by normalizing a denormalized number's fraction field and subtracting from the exponent the number of shifts required to do so. The exponent is encoded as a two's complement negative number.

| | |
|---|-----------|
| General Information | 1 |
| SN74ACT8818A 16-Bit Microsequencer | 2 |
| SN74ACT8832A 32-Bit Registered ALU | 3 |
| SN74ACT8836A 32- × 32-Bit Parallel Multiplier | 4 |
| SN74ACT8841A Digital Crossbar Switch | 5 |
| SN74ACT8847 64-Bit Floating-Point/Integer Unit | 6 |
| SN74ACT8847 Application Information | 7 |
| SN74ACT8867 32-Bit Vector Processor Unit | 8 |
| Design Support | 9 |
| Mechanical Data | 10 |

1 General Information

2 SN74ACT8015A 16 Bit Multiplexer

3 SN74ACT8024 32 Bit Register A/D

4 SN74ACT8080A 32 x 8 Bit Parallel Memory

5 SN74ACT80A1A Digital Display Driver

6 SN74ACT847 64 Bit Floating-Point Register Unit

7 SN74ACT8847 Application Information

8 SN74ACT8903 32 Bit Vector Processor Unit

9 Design Support

10 Attachment Date

Contents

| | <i>Page</i> |
|---|-------------|
| Sum of Products and Product of Sums | 7-9 |
| Matrix Operations | 7-12 |
| Representation of Variables | 7-12 |
| Sample Matrix Transformation | 7-13 |
| Microinstructions for Sample Matrix Manipulation | 7-19 |
| Chebyshev Routines for the SN74ACT8847 FPU | 7-23 |
| Introduction | 7-23 |
| Overview of Chebyshev's Expansion Method | 7-24 |
| Format for the Remainder of the Application Note | 7-26 |
| Cosine Routine Using Chebyshev's Method | 7-27 |
| Steps Required to Perform the Calculation | 7-27 |
| Algorithms for the Three Steps | 7-28 |
| Required System Intervention | 7-29 |
| Number of 'ACT8847 Cycles Required to | |
| Calculate Cosine(x) | 7-29 |
| Listing of the Chebyshev Constants (c's) | 7-29 |
| Pseudocode Table for the Cosine(x) Calculation | 7-30 |
| Microcode Table for the Cosine(x) Calculation | 7-33 |
| Sine Routine Using Chebyshev's Method | 7-35 |
| Steps Required to Perform the Calculation | 7-35 |
| Algorithms for the Three Steps | 7-35 |
| Required System Intervention | 7-36 |
| Number of 'ACT8847 Cycles Required to | |
| Calculate Sine(x) | 7-36 |
| Listing of the Chebyshev Constants (c's) | 7-36 |
| Pseudocode Table for the Sine(x) Calculation | 7-37 |
| Microcode Table for the Sine(x) Calculation | 7-40 |
| Tangent Routine Using Chebyshev's Method | 7-42 |
| Steps Required to Perform the Calculation | 7-42 |
| Algorithms for the Three Steps | 7-42 |
| Required System Intervention | 7-44 |

Contents (Continued)

| | Page |
|--|-------------|
| Number of 'ACT8847 Cycles Required to Calculate | |
| Tangent(x) | 7-44 |
| Listing of the Chebyshev Constants (c's) | 7-44 |
| Pseudocode Table for the Tangent(x) Calculation | 7-45 |
| Microcode Table for the Tangent(x) Calculation | 7-49 |
| ArcSine and ArcCosine Routine Using Chebyshev's | |
| Method | 7-53 |
| Steps Required to Perform the Calculation | 7-53 |
| Algorithms for the Three Steps | 7-53 |
| Required System Intervention | 7-55 |
| Number of 'ACT8847 Cycles Required to Calculate | |
| ArcSine(x) and ArcCosine(x) | 7-55 |
| Listing of the Chebyshev Constants (c's) | 7-55 |
| Pseudocode Table for the ArcSine(x) and ArcCosine | |
| Calculation | 7-56 |
| Microcode Table for the ArcSine(x) and ArcCosine(x) | |
| Calculation | 7-59 |
| ArcTangent Routine Using Chebyshev's Method | 7-62 |
| Steps Required to Perform the Calculation | 7-62 |
| Algorithms for the Three Steps | 7-63 |
| Required System Intervention | 7-64 |
| Number of 'ACT8847 Cycles Required to Calculate | |
| Arctangent(x) | 7-64 |
| Listing of the Chebyshev Constants (c's) | 7-65 |
| Pseudocode Table for the ArcTangent(x) Calculation | 7-66 |
| Microcode Table for the ArcTangent(x) Calculation | 7-71 |
| Exponential Routine Using Chebyshev's Method | 7-75 |
| Steps Required to Perform the Calculation | 7-75 |
| Algorithms for the Three Steps | 7-76 |
| Required System Intervention | 7-77 |
| Number of 'ACT8847 Cycles Required to | |
| Calculate Exp(x) | 7-77 |
| Listing of the Chebyshev Constants (c's) | 7-77 |
| Pseudocode Table for the Exp(x) Calculation | 7-78 |
| Microcode Table for the Exp(x) Calculation | 7-81 |
| High-Speed Vector Math and 3-D Graphics | 7-84 |
| Introduction | 7-84 |
| SN74ACT8847 Floating-Point Unit | 7-84 |

Contents (Continued)

| | <i>Page</i> |
|--|-------------|
| Mathematical Processing Applications | 7-87 |
| Graphics Applications | 7-87 |
| Vector Arithmetic | 7-88 |
| Computational Operations on Data Vectors | 7-88 |
| Compare Operations on Data Vectors | 7-96 |
| Graphics Applications | 7-100 |
| Creating a 3-D Image | 7-100 |
| Three-Dimensional Coordinate Transforms | 7-104 |
| Three-Dimensional Clipping | 7-107 |
| Summary of Graphics Systems Performance | 7-119 |

List of Illustrations

| <i>Figure</i> | | <i>Page</i> |
|---------------|---|-------------|
| 1 | Sequence of Matrix Operations | 7-15 |
| 2 | Resultant Matrix Transformation | 7-22 |
| 3 | SN74ACT8847 Floating-Point Unit | 7-86 |
| 4 | Creating a 3-D Image | 7-102 |
| 5 | View Volume | 7-103 |
| 6(a) | Model of Procedure for Creating a 3-D Graphic | 7-104 |
| 6(b) | Model of Creating and Transforming a 3-D Graphic | 7-104 |
| 7 | Viewing Pyramid Showing Six Clipping Planes | 7-108 |

Contents (Continued)

| | |
|-------|--|
| 7-118 | Summary of Graphics Systems Performance |
| 7-107 | Three-Dimensional Clipping |
| 7-104 | Three-Dimensional Coordinate Transforms |
| 7-100 | Creating a 3-D Image |
| 7-100 | Graphics Applications |
| 7-98 | Compare Operations on Data Vectors |
| 7-88 | Computational Operations on Data Vectors |
| 7-88 | Vector Arithmetic |
| 7-87 | Graphics Applications |
| 7-87 | Mathematical Processing Applications |

List of Illustrations

| | | |
|-------|--|------|
| 7-108 | Viewing Pyramid Showing Six Clipping Planes | 7 |
| 7-104 | Model of Creating and Transforming a 3-D Graphic | 8(b) |
| 7-104 | Model of Procedure for Creating a 3-D Graphic | 8(a) |
| 7-103 | View Volume | 5 |
| 7-102 | Creating a 3-D Image | 4 |
| 7-88 | SN74ACT8847 Floating-Point Unit | 3 |
| 7-22 | Resultant Matrix Transformation | 2 |
| 7-15 | Sequence of Matrix Operations | 1 |



List of Tables

| <i>Table</i> | <i>Page</i> |
|--|-------------|
| 1 Pseudocode for Fully Pipelined Double-Precision Sum of Products (CLKMODE = 0, CONFIG1-CONFIG0 = 10, PIPES2-PIPES0 = 000) | 7-10 |
| 2 Pseudocode for Fully Pipelined Double-Precision Product of Sums (CLKMODE = 0, CONFIG1-CONFIG0 = 10, PIPES2-PIPES0 = 000) | 7-11 |
| 3 Single-Precision Matrix Multiplication (PIPES2-PIPES0 = 010) | 7-20 |
| 4 Microinstructions for Sample Matrix Multiplication | 7-21 |
| 5 Fully Pipelined Single-Precision Sum of Products (PIPES2-PIPES0 = 000) | 7-23 |
| 6 Cycle Count and Execution Speed for the Seven Chebyshev Functions | 7-24 |
| 7 Pseudocode for Chebyshev Cosine Routine (PIPES2-PIPES0 = 010, RND1-RND0 = 00) | 7-30 |
| 8 Pseudocode for Chebyshev Sine Routine (PIPES2-PIPES0 = 010, RND1-RND0 = 00) | 7-37 |
| 9 Pseudocode for Chebyshev Tangent Routine (PIPES2-PIPES0 = 010, RND1-RND0 = 00) | 7-45 |
| 10 Pseudocode for Chebyshev ArcSine and ArcCosine Routine (PIPES2-PIPES0 = 010, RND1-RND0 = 00) | 7-56 |
| 11 Pseudocode for Chebyshev ArcTangent Routine (PIPES2-PIPES0 = 010, RND1-RND0 = 00) | 7-66 |
| 12 Pseudocode for Chebyshev Exponential Routine (PIPES2-PIPES0 = 010, RND1-RND0 = 00) | 7-78 |
| 13 Data Flow for Pipelined Single-Precision Vector Add, N = 6 | 7-89 |
| 14 Program Listing for Pipelined Single-Precision Vector Add, N = 6 | 7-89 |
| 15 Data Flow for Pipelined Single-Precision Vector Multiply, N = 6 | 7-90 |
| 16 Program Listing for Pipelined Single-Precision Vector Multiply, N = 6 | 7-90 |
| 17 Data Flow for Unpipelined Single-Precision Vector Multiply, N = 6 | 7-91 |
| 18 Data Flow for Pipelined Single-Precision Sum of Products, N = 8 | 7-92 |

List of Tables (Continued)

| <i>Table</i> | <i>Page</i> |
|--|-------------|
| 19 Program Listing for Pipelined Single-Precision Sum of Products, $N = 8$ | 7-93 |
| 20 Data Flow for Unpipd Single-Precision Sum of Products, $N = 8$ | 7-93 |
| 21 Data Flow for 'ACT8847 Pipelined Single-Precision Vector Divide | 7-95 |
| 22 Program Listing for 'ACT8847 Pipelined Single-Precision Vector Divide | 7-95 |
| 23 Data Flow for Pipelined Single-Precision Vector MAX | 7-96 |
| 24 Data Flow for Pipelined Single-Precision Interleaved Vector MAX/MIN | 7-97 |
| 25 Program Listing for Pipelined Single-Precision Interleaved Vector MAX/MIN | 7-98 |
| 26 Data Flow for Unpipd Single-Precision Vector MAX | 7-98 |
| 27 Data Flow for Pipelined Single-Precision List MAX | 7-99 |
| 28 Program Listing for Pipelined Single-Precision List MAX ... | 7-100 |
| 29 Partial Data Flow for Product of $[X, Y, Z, W]$ and General Transform Matrix | 7-105 |
| 30 Partial Data Flow for Product of $[X, Y, Z, W]$ and Reduced Transform Matrix | 7-106 |
| 31 Data Flow for Clipping a Line Segment Against the $Z = N$ Plane | 7-110 |
| 32 Data Flow for Computing t_1, t_2, s_1 , and s_2 Using an SN74ACT8847 | 7-112 |
| 33 Program Listing for Three-Processor Clip to Compute t_1, t_2, s_1 , and s_2 | 7-113 |
| 34 $A > B$ Comparison Function Table | 7-114 |
| 35 Data Flow for Accept/Reject Testing | 7-115 |
| 36 Data Flow for the X Processor | 7-117 |
| 37 Program Listing for the X Processor | 7-118 |
| 38 Summary of Graphics Systems Performance | 7-119 |
| 39 Available Options for Graphic System Designs | 7-119 |

Table 1. Pseudocode for Fully Pipelined Double-Precision Sum of Products[†]
(CLKMODE = 0, CONFIG1-CONFIG0 = 10, PIPES2-PIPES0 = 000)

| CLK | DA BUS | DB BUS | TEMP REG | INS BUS | INS REG | RA REG | RB REG | MUL PIPE | P REG | C REG | ALU PIPE | S REG | Y BUS |
|---|-----------|-----------|-------------|--------------------|---------------------|-----------|-----------|-------------|----------|----------|-------------|----------|----------|
|  1 | A1 MSH | B1 MSH | A1,B1MSH | A1 * B1 | | | | | | | | | |
|  2 | A1 LSH | B1 LSH | A1,B1LSH | A1 * B1 | A1 * B1 | A1 | B1 | | | | | | |
|  3 | A2 MSH | B2 MSH | A2,B2MSH | A2 * B2 | A1 * B1 | A1 | B1 | A1 * B1 | | | | | |
|  4 | A2 LSH | B2 LSH | A2,B2LSH | A2 * B2 | A2 * B2 | A2 | B2 | A1 * B1 | | | | | |
|  5 | A3 MSH | B3 MSH | A3,B3MSH | PR + CR A3 * B3 | A2 * B2 | A2 | B2 | A2 * B2 | P1 | | | | |
|  6 | A3 LSH | B3 LSH | A3,B3LSH | PR + CR A3 * B3 | PR + CR, A3 * B3 | A3 | B3 | A2 * B2 | P1 | P1 | | | |
|  7 | A4 MSH | B4 MSH | A4,B4MSH | PR + SR A4 * B4 | PR + SR, A3 * B3 | A3 | B3 | A3 * B3 | P2 | P1 | P1 + P1 | | |
|  8 | A4 LSH | B4 LSH | A4,B4LSH | PR + SR A4 * B4 | PR + SR, A4 * B4 | A4 | B4 | A3 * B3 | P2 | P1 | P1 + P2 | | |
|  9 | A5 MSH | B5 MSH | A5,B5MSH | PR + SR A5 * B5 | PR + SR, A4 * B4 | A4 | B4 | A4 * B4 | P3 | P2 | S1 + P2 | S1 | |
|  10 | A5 LSH | B5 LSH | A5,B5LSH | PR + SR A5 * B5 | PR + SR, A5 * B5 | A5 | B5 | A4 * B4 | P3 | P2 | S1 + P3 | S1 | |
|  11 | A6 MSH | B6 MSH | A6,B6MSH | PR + SR A6 * B6 | PR + SR, A5 * B5 | A5 | B5 | A5 * B5 | P4 | P2 | XXXXXX | S2 | |
|  12 | | | | | | | | | P4 | P2 | | S2 | |

[†]PR = Product Register
 SR = Sum Register
 CR = Constant (C) Register

Table 2. Pseudocode for Fully Pipelined Double-Precision Product of Sums[†]
(CLKMODE=0, CONFIG1-CONFIG0=10, PIPES2-PIPES0=000)

| CLK | DA BUS | DB BUS | TEMP REG | INS BUS | INS REG | RA REG | RB REG | MUL PIPE | P REG | C REG | ALU PIPE | S REG | Y BUS |
|--|-----------|-----------|-------------|----------------|----------------|--------------|--------------|-------------|----------|--------------|-------------|----------|----------|
|  1 | A1MSH | B1MSH | A1,B1MSH | A1+B1 | | | | | | | | | |
|  2 | A1LSH | B1LSH | A1,B1LSH | A1+B1 | A1+B1 | A1 | B1 | | | | | | |
|  3 | A2MSH | B2MSH | A2,B2MSH | A2+B2 | A1+B1 | A1 | B1 | | | | A1+B1 | | |
|  4 | A2LSH | B2LSH | A2,B2LSH | A2+B2 | A2+B2 | A2 | B2 | | | | A1+B1 | S1 | |
|  5 | A3MSH | B3MSH | A3,B3MSH | CR*SR A3+B3 | A2+B2 | A2 | B2 | | | ENRC=0 S1 | A2+B2 | S1 | |
|  6 | A3LSH | B3LSH | A3,B3LSH | CR*SR A3+B3 | CR*SR A3+B3 | A3 | B3 | | | S1 | A2+B2 | S2 | |
|  7 | XXX | XXX | XXX | NOP | CR*SR A3+B3 | A3 | B3 | S1*S2 | | S1 | A3+B3 | S2 | |
|  8 | A4MSH | B4MSH | A4,B4MSH | PR*SR A4+B4 | NOP | ENRA=0 A3 | ENRB=0 B3 | S1*S2 | | S1 | | XXX | |
|  9 | A4LSH | B4LSH | A4,B4LSH | PR*SR A4+B4 | PR*SR A4+B4 | A4 | B4 | XXX | P1 | S1 | XXX | S3 | |
|  10 | XXX | XXX | XXX | NOP | PR*SR A4+B4 | A4 | B4 | P1*S3 | P1 | S1 | A4+B4 | S3 | |
|  11 | A5MSH | B5MSH | A5,B5MSH | PR*SR A5+B5 | NOP | ENRA=0 A4 | ENRB=0 B4 | P1*S3 | XXX | S1 | A4+B4 | XXX | |
|  12 | A5LSH | B5LSH | A5,B5LSH | PR*SR A5+B5 | PR*SR A5+B5 | A5 | B5 | XXX | P2 | S1 | X | S4 | |

NOTE: NOP instruction is 011 0000 0000.

[†]PR = Product Register

SR = Sum Register

CR = Constant (C) Register

Matrix Operations

The 'ACT8847 floating-point unit can also be used to perform matrix manipulations involved in graphics processing or digital signal processing. The FPU multiplies and adds data elements, executing sequences of microprogrammed calculations to form new matrices.

Representation of Variables

In state representations of control systems, an n-th order linear differential equation with constant coefficients can be represented as a sequence of n first-order linear differential equations expressed in terms of state variables:

$$\frac{dx_1}{dt} = x_2, \dots, \frac{dx_{(n-1)}}{dt} = x_n$$

For example, in vector-matrix form the equations of an nth-order system can be represented as follows:

$$\frac{d}{dt} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_{11} & \dots & b_{1n} \\ \vdots & & \vdots \\ \vdots & & \vdots \\ b_{n1} & \dots & b_{nn} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}$$

$$\text{or, } \dot{X} = a_x + b_u$$

Expanding the matrix equation for one state variable, dx_1/dt , results in the following expression:

$$\dot{X}_1 = (a_{11} * x_1 + \dots + a_{1n} * x_n) + (b_{11} * u_1 + \dots + b_{1n} * u_n)$$

where $\dot{X}_1 = dx_1/dt$.

Sequences of multiplications and additions are required when such state space transformations are performed, and the 'ACT8847 has been designed to support such sum-of-products operations. An $n \times n$ matrix A multiplied by an $n \times n$ matrix X yields an $n \times n$ matrix C whose elements c_{ij} are given by this equation:

$$c_{ij} = \sum_{k=1}^n a_{ik} * x_{kj} \quad \text{for } i=1, \dots, n \quad j=1, \dots, n \quad (1)$$

For the c_{ij} elements to be calculated by the 'ACT8847, the corresponding elements a_{jk} and x_{kj} must be stored outside the 'ACT8847 and fed to the 'ACT8847 in the proper order required to effect a matrix multiplication such as the state space system representation just discussed.

Sample Matrix Transformation

The matrix manipulations commonly performed in graphics systems can be regarded as geometrical transformations of graphic objects. A matrix operation on another matrix representing a graphic object may result in scaling, rotating, transforming, distorting, or generating a perspective view of the image. By performing a matrix operation on the position vectors which define the vertices of an image surface, the shape and position of the surface can be manipulated.

The generalized 4×4 matrix for transforming a three-dimensional object with homogeneous coordinates is shown below:

$$T = \begin{bmatrix} a & b & c & : & d \\ e & f & g & : & h \\ i & j & k & : & l \\ \cdot & \cdot & \cdot & : & \cdot \\ m & n & o & : & p \end{bmatrix}$$

The matrix T can be partitioned into four component matrices, each of which produces a specific effect on the resultant image:

$$\begin{bmatrix} \cdot & \cdot & \cdot & : & \cdot \\ \cdot & \cdot & \cdot & : & \cdot \\ \cdot & \cdot & \cdot & : & \cdot \\ 3 \times 3 & : & x & & \\ \cdot & \cdot & \cdot & : & \cdot \\ 1 \times 3 & : & 1 \times 1 \end{bmatrix}$$

The 3×3 matrix produces linear transformation in the form of scaling, shearing and rotation. The 1×3 row matrix produces translation, while the 3×1 column matrix produces perspective transformation with multiple vanishing points. The final single element 1×1 produces overall scaling. Overall operation of the transformation matrix T on the position vectors of a graphic object produces a combination of shearing, rotation, reflection, translation, perspective, and overall scaling.

The rotation of an object about an arbitrary axis in a three-dimensional space can be carried out by first translating the object such that the desired axis of rotation passes through the origin of the coordinate system, then rotating the object about the axis

The diagram illustrates the composition of two transformations, R and S , resulting in a rotation RS .

Transformation R (Reflection across $y=x$):

| | | | |
|----|----|----|---|
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| -a | -b | -c | 1 |

Transformation S (Reflection across $y=-x$):

| | | | |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| a | b | c | 1 |

Composition RS (Rotation by 90° counter-clockwise):

| | | | |
|----|----|----|---|
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| -a | -b | -c | 1 |

translation
back to initial
position

$$n_1 n_2 (1 - \cos \phi) +$$

$$\begin{bmatrix} n_1 n_3 (1 - \cos \phi) + n_2 \sin \phi & n_2 n_3 (1 - \cos \phi) - n_1 \sin \phi & n_3^2 + (1 - n_3)^2 \cos \phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

 $+q32)^{1/2} = \text{direction}$ $+q32)^{1/2} = \text{direction}$ $+q_3^2)^{1/2} = \text{direction}$
$$\bar{n} = (n_1 \ n_2 \ n_3) = \text{unit vector for } \bar{Q}$$

\vec{Q} = vector defining axis of rotation = $[q_1 \ q_2 \ q_3]$

ϕ = the rotation angle about \bar{Q}

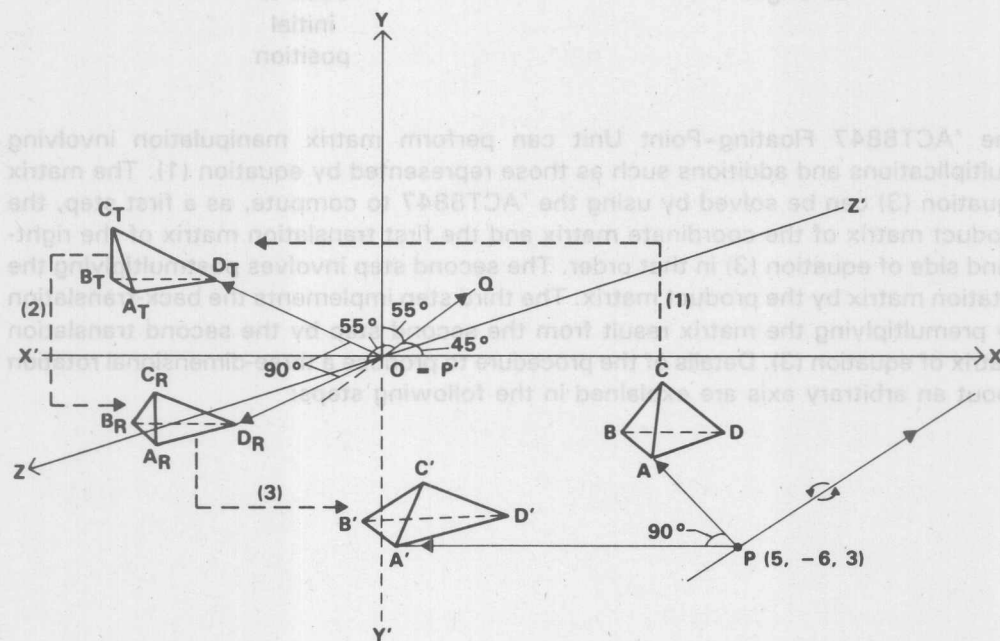
A general rotation using equation (2) is effected by determining the $[x \ y \ z]$ coordinates

define matrix [R]. Suppose, for example, that a tetrahedron ABCD, represented by the coordinate matrix below is to be rotated about an axis of rotation RX which passes through a point P = [5 -6 3 1] and whose direction cosines are given by unit vector [n1 = 0.866, n2 = 0.5, n3 = 0.707]. The angle of rotation θ is 90 degrees (see Figure 1). The rotation matrix [R] becomes

| | | | |
|---|----|---|---|
| 2 | -3 | 3 | 1 |
| 1 | -2 | 2 | 1 |
| 2 | -1 | 2 | 1 |
| 2 | -2 | 2 | 1 |

A
B
C
D

$$R = \begin{bmatrix} 0.750 & 1.140 & 0.112 & 0 \\ -0.274 & 0.250 & 1.220 & 0 \\ 1.112 & -0.513 & 0.500 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



- (1) THIS ARROW DEPICTS THE FIRST TRANSLATION
- (2) THIS ARROW DEPICTS THE 90° ROTATION
- (3) THIS ARROW DEPICTS THE BACK TRANSLATION

Figure 1. Sequence of Matrix Operations

The point transformation equation (2) can be expanded to include all the vertices of the tetrahedron as follows:

$$\begin{bmatrix} x_a & y_a & z_a & h_1 \\ x_b & y_b & z_b & h_2 \\ x_c & y_c & z_c & h_3 \\ x_d & y_d & z_d & h_4 \end{bmatrix} = \begin{bmatrix} 2 & -3 & 3 & 1 \\ 1 & -2 & 2 & 1 \\ 2 & -1 & 2 & 1 \\ 2 & -2 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -5 & 6 & -3 & 1 \end{bmatrix} \begin{bmatrix} 0.750 & 1.140 & 0.112 & 0 \\ -0.274 & 0.250 & 1.22 & 0 \\ 1.112 & -0.513 & 0.500 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 5 & -6 & 3 & 1 \end{bmatrix} \quad (3)$$

translation to origin
rotation about origin
translation back to initial position

The 'ACT8847 Floating-Point Unit can perform matrix manipulation involving multiplications and additions such as those represented by equation (1). The matrix equation (3) can be solved by using the 'ACT8847 to compute, as a first step, the product matrix of the coordinate matrix and the first translation matrix of the right-hand side of equation (3) in that order. The second step involves postmultiplying the rotation matrix by the product matrix. The third step implements the back-translation by premultiplying the matrix result from the second step by the second translation matrix of equation (3). Details of the procedure to produce a three-dimensional rotation about an arbitrary axis are explained in the following steps:

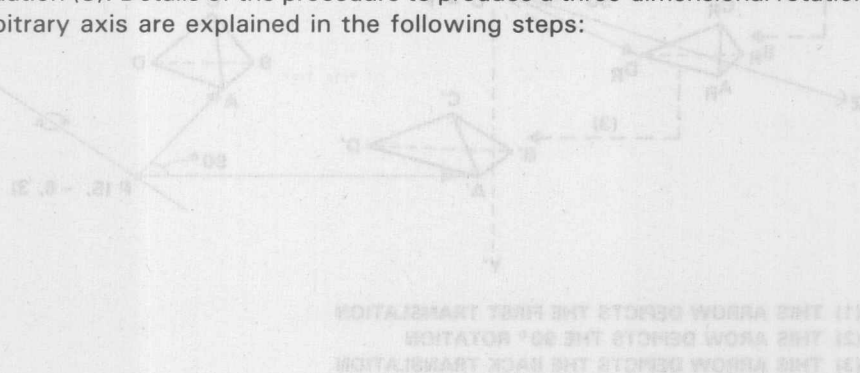


Figure 7. Sequence of Matrix Operations

Step 1

Translate the tetrahedron so that the axis of rotation passes through the origin. This process can be accomplished by multiplying the coordinate matrix by the translation matrix as follows:

$$\begin{bmatrix} 2 & -3 & 3 & 1 \\ 1 & -2 & 2 & 1 \\ 2 & -1 & 2 & 1 \\ 2 & -2 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -5 & 6 & -3 & 1 \end{bmatrix} = \begin{bmatrix} (2-5) & (-3+6) & (3-3) & 1 \\ (1-5) & (-2+6) & (2-3) & 1 \\ (2-5) & (-1+6) & (2-3) & 1 \\ (2-5) & (-2+6) & (2-3) & 1 \end{bmatrix}$$

translation
vertices of translated
to origin
tetrahedron

$$= \begin{bmatrix} -3 & +3 & 0 & 1 \\ -4 & +4 & -1 & 1 \\ -3 & +5 & -1 & 1 \\ -3 & +4 & -1 & 1 \end{bmatrix} \begin{matrix} \text{---} \text{ AT} \\ \text{---} \text{ BT} \\ \text{---} \text{ CT} \\ \text{---} \text{ DT} \end{matrix}$$

The 'ACT8847 could compute the translated coordinates AT, BT, CT, DT as indicated above. However, an alternative method resulting in a more compact solution is presented below.

Step 2

Rotate the tetrahedron about the axis of rotation which passes through the origin after the translation of Step 1. To implement the rotation of the tetrahedron, postmultiply the rotation matrix [R] by the translated coordinate matrix from Step 1. The resultant matrix represents the rotated coordinates of the tetrahedron about the origin as follows:

$$\begin{bmatrix} -3 & 3 & 0 & 1 \\ -4 & 4 & -1 & 1 \\ -3 & 5 & -1 & 1 \\ -3 & 4 & -1 & 1 \end{bmatrix} \begin{bmatrix} 0.750 & 1.140 & 0.112 & 0 \\ -0.274 & 0.250 & 1.22 & 0 \\ 1.112 & -0.513 & 0.500 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -3.072 & -2.670 & 3.324 & 1 \\ -5.208 & -3.047 & 3.932 & 1 \\ -4.732 & -1.657 & 5.264 & 1 \\ -4.458 & -1.907 & 4.044 & 1 \end{bmatrix}$$

rotation about origin
rotated coordinates

Step 3

Translate the rotated tetrahedron back to the original coordinate space. This is done by premultiplying the resultant matrix of Step 2 by the translation matrix. The following calculations produces the final coordinate matrix of the transformed object:

$$\begin{bmatrix} -3.072 & -2.670 & 3.324 & 1 \\ -5.208 & -3.047 & 3.932 & 1 \\ -4.732 & -1.657 & 5.264 & 1 \\ -4.458 & -1.907 & 4.044 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 5 & -6 & 3 & 1 \end{bmatrix} = \begin{bmatrix} 1.928 & -8.670 & 6.324 & 1 \\ -0.208 & -9.047 & 6.932 & 1 \\ 0.268 & -7.657 & 8.264 & 1 \\ 0.542 & -7.907 & 7.044 & 1 \end{bmatrix}$$

translate back

final rotated coordinates

A more compact solution to these transformation matrices is a product matrix that combines the two translation matrices and the rotation matrix in the order shown in equation (3). Equation (3) will then take the following form:

$$\begin{bmatrix} x_a & y_a & z_a & h_1 \\ x_b & y_b & z_b & h_2 \\ x_c & y_c & z_c & h_3 \\ x_d & y_d & z_d & h_4 \end{bmatrix} =$$

$$\begin{bmatrix} 2 & -3 & 3 & 1 \\ 1 & -2 & 2 & 1 \\ 2 & -1 & 2 & 1 \\ 2 & -2 & 2 & 1 \end{bmatrix} \begin{bmatrix} 0.750 & 1.140 & 0.112 & 0 \\ -0.274 & 0.250 & 1.220 & 0 \\ 1.112 & -0.513 & 0.500 & 0 \\ -3.730 & -8.661 & 8.260 & 1 \end{bmatrix}$$

transformation matrix

The newly transformed coordinates resulting from the postmultiplication of the transformation matrix by the coordinate matrix of the tetrahedron can be computed using equation (1) which was cited previously:

$$c_{ij} = \sum_{k=1}^n a_{ik} * x_{kj} \quad \text{for } i=1, \dots, n \quad j=1, \dots, n \quad (1)$$

For example, the coordinates may be computed as follows:

$$\begin{aligned} x_a = c_{11} &= a_{11} * x_{11} + a_{12} * x_{21} + a_{13} * x_{31} + a_{14} * x_{41} \\ &= 2 * 0.750 + (-3) * (-0.274) + 3 * 1.112 + 1 * (-3.73) \\ &= 1.5 + 0.822 + 3.336 - 3.73 \\ &= 1.928 \end{aligned}$$

$$\begin{aligned} y_a = c_{12} &= a_{11} * x_{12} + a_{12} * x_{22} + a_{13} * x_{32} + a_{14} * x_{42} \\ &= 2 * 1.140 + (-3) * 0.250 + 3 * (-0.513) + 1 * (-8.661) \\ &= 2.28 - 0.75 - 1.539 - 8.661 \\ &= -8.67 \end{aligned}$$

$$\begin{aligned} z_a = c_{13} &= a_{11} * x_{13} + a_{12} * x_{23} + a_{13} * x_{33} + a_{14} * x_{43} \\ &= 2 * 0.112 + (-3) * 1.220 + 3 * 0.500 + 1 * 8.260 \\ &= 0.224 - 3.66 + 1.5 + 8.260 \\ &= 6.324 \end{aligned}$$

$$\begin{aligned} h_1 = c_{14} &= a_{11} * x_{14} + a_{12} * x_{24} + a_{13} * x_{34} + a_{14} * x_{44} \\ &= 2 * 0 + (-3) * 0 + 3 * 0 + 1 * 1 \\ &= 0 + 0 + 0 + 1 \\ &= 1 \end{aligned}$$

$$\text{---} \quad A' = [1.928 \quad -8.67 \quad 6.324 \quad 1]$$

The other rotated vertices are computed in a similar manner:

$$B' = [-5.208 \quad -3.047 \quad 3.932 \quad 1]$$

$$C' = [-4.732 \quad -1.657 \quad 5.264 \quad 1]$$

$$D' = [-4.458 \quad -1.907 \quad 4.044 \quad 1]$$

Microinstructions for Sample Matrix Manipulation

The 'ACT8847 FPU can compute the coordinates for graphic objects over a broad dynamic range. Also, the homogeneous scalar factors h_1 , h_2 , h_3 and h_4 may be made unity due to the availability of large dynamic range. In the example presented below, some of the calculations pertaining to vertex A' are shown but the same approach can be applied to any number of points and any vector space.

The calculations below show the sequence of operations for generating two coordinates, x_a and y_a , of the vertex A' after rotation. The same sequence could be continued to generate the remaining two coordinates for A' (z_a and h_1). The other vertices of the tetrahedron, B' , C' , and D' , can be calculated in a similar way.

Table 3 presents a pseudocode description of the operations, clock cycles, and register contents for a single-precision matrix multiplication using the sum-of-products sequence presented in an earlier section. Registers used include the RA and RB input registers and the product (P) and sum (S) registers.

Table 3. Single-Precision Matrix Multiplication (PIPES2-PIPES0 = 010)

| CLOCK CYCLE | MULTIPLIER/ALU OPERATIONS | PSEUDOCODE |
|-------------|---|--|
| 1 | Load a11, x11 SP Multiply | $a_{11} \rightarrow RA, x_{11} \rightarrow RB$ $p_1 = a_{11} * x_{11}$ |
| 2 | Load a12, x21 SP Multiply Pass P to S | $a_{12} \rightarrow RA, x_{21} \rightarrow RB$ $p_2 = a_{12} * x_{21}$ $p_1 \rightarrow P(p_1)$ |
| 3 | Load a13, x31 SP Multiply Add P to S | $a_{13} \rightarrow RA, x_{31} \rightarrow RB$ $p_3 = a_{13} * x_{31}, p_2 \rightarrow P(p_2)$ $P(p_1) + 0 \rightarrow S(p_1)$ |
| 4 | Load a14, x41 SP Multiply Add P to S | $a_{14} \rightarrow RA, x_{41} \rightarrow RB$ $p_4 = a_{14} * x_{41}, p_3 \rightarrow P(p_3)$ $P(p_2) + S(p_1) \rightarrow S(p_1 + p_2)$ |
| 5 | Load a11, x12 SP Multiply Add P to S | $a_{11} \rightarrow RA, x_{12} \rightarrow RB$ $p_5 = a_{11} * x_{12}, p_4 \rightarrow P(p_4)$ $P(p_3) + S(p_1 + p_2) \rightarrow S(p_1 + p_2 + p_3)$ |
| 6 | Load a12, x22 SP Multiply Pass P to S Output S | $a_{12} \rightarrow RA, x_{22} \rightarrow RB$ $p_6 = a_{12} * x_{22}, p_5 \rightarrow P(p_5)$ $P(p_4) + S(p_1 + p_2 + p_3) \rightarrow$ $S(p_1 + p_2 + p_3 + p_4)$ |
| 7 | Load a13, x32 SP Multiply Add P to S | $a_{13} \rightarrow RA, x_{32} \rightarrow RB$ $p_7 = a_{13} * x_{32}, p_6 \rightarrow P(p_6)$ $P(p_5) + 0 \rightarrow S(p_5)$ |
| 8 | Load a14, x42 SP Multiply Add P to S | $a_{14} \rightarrow RA, x_{42} \rightarrow RB$ $p_8 = a_{14} * x_{42}, p_7 \rightarrow P(p_7)$ $P(p_6) + S(p_5) \rightarrow S(p_5 + p_6)$ |
| 9 | Next operands Next instruction Add P to S | $A \rightarrow RA, B \rightarrow RB$ $p_i = A * B, p_8 \rightarrow P(p_8)$ $P(p_7) + S(p_5 + p_6) \rightarrow S(p_5 + p_6 + p_7)$ |
| 10 | Next operands Next instruction Output S | $C \rightarrow RA, D \rightarrow RB$ $p_j = C * D, p_i \rightarrow P(p_i)$ $P(p_8) + S(p_5 + p_6 + p_7) \rightarrow$ $S(p_5 + p_6 + p_7 + p_8)$ |

Table 4. Microinstructions for Sample Matrix Multiplication

Calculations for vertices B', C', and D', can be executed in 48 cycles, 16 cycles for each vertex. Processing time improves when the transformation matrix is reduced, i.e., when the last column has the form shown below:

| |
|---|
| 0 |
| 0 |
| 0 |
| 1 |

7-21

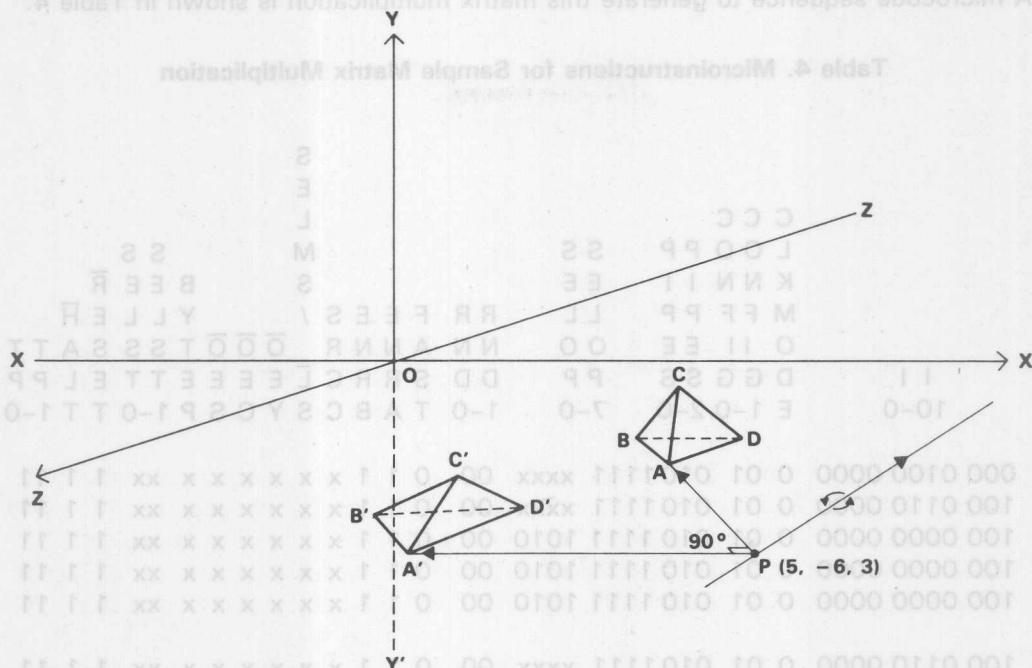


Figure 2. Resultant Matrix Transformation

This microprogram can also be written to calculate sums of products with all pipeline registers enabled so that the FPU can operate in its fastest mode. Because of timing relationships, the C register is used in some steps to hold the intermediate sum of products. Latency due to pipelining and chained data manipulation is 11 cycles for calculation of the first coordinate, and four cycles each for the other three coordinates.

After calculation of the first vertex, 16 cycles are required to calculate the four coordinates of each subsequent vertex. Table 5 presents the sequence of calculations for the first two coordinates, x_A and y_A .

Products in Table 5 are numbered according to the clock cycle in which the operands and instruction were loaded into the RA, RB, and I register, and execution of the instruction began. Sums indicated in Table 5 are listed below:

$$\begin{array}{lll}
 s1 = p1 + 0 & s5 = p5 + p7 & s9 = p10 + p12 \\
 s2 = p1 + p3 & s6 = p6 + p8 & xA = p1 + p2 + p3 + p4 \\
 s3 = p2 + p4 & s7 = p9 + 0 & yA = p5 + p6 + p7 + p8 \\
 s4 = p5 + 0 & s8 = p9 + p11 &
 \end{array}$$

Table 5. Fully Pipelined Single-Precision Sum of Products (PIPES2-PIPES0 = 000)

| CLOCK CYCLE | I BUS | DA BUS | DB BUS | I REG | RA REG | RB REG | MUL PIPE | ALU PIPE | P REG | S REG | C REG | Y BUS |
|-------------|-------|--------|--------|-------|--------|--------|----------|----------|-------|-------|-------|-------|
| 0 | Mul | x11 | a11 | | | | | | | | | |
| 1 | Mul | x21 | a12 | Mul | x11 | a11 | | | | | | |
| 2 | Chn | x31 | a13 | Mul | x21 | a12 | p1 | | | | | |
| 3 | Mul | x41 | a14 | Chn | x31 | a13 | p2 | | p1 | | | |
| 4 | Chn | x12 | a11 | Mul | x41 | a14 | p3 | s1 | p2 | | | |
| 5 | Chn | x22 | a12 | Chn | x12 | a11 | p4 | † | p3 | s1 | p2 | |
| 6 | Chn | x32 | a13 | Chn | x22 | a12 | p5 | s2 | p4 | † | p2 | |
| 7 | Chn | x42 | a14 | Chn | x32 | a13 | p6 | s3 | p5 | s2 | p2 | |
| 8 | Chn | x13 | a11 | Chn | x42 | a14 | p7 | s4 | p6 | s3 | s2 | |
| 9 | Chn | x23 | a12 | Chn | x13 | a11 | p8 | xA | p7 | s4 | p6 | |
| 10 | Chn | x33 | a13 | Chn | x23 | a12 | p9 | s5 | p8 | xA | p6 | xA |
| 11 | Chn | x43 | a14 | Chn | x33 | a13 | p10 | s6 | p9 | s5 | p6 | |
| 12 | Chn | x14 | a11 | Chn | x43 | a14 | p11 | s7 | p10 | s6 | s5 | |
| 13 | Chn | x24 | a12 | Chn | x14 | a11 | p12 | yA | p11 | s7 | p10 | |
| 14 | Chn | x34 | a13 | Chn | x24 | a12 | p13 | s8 | p12 | yA | p10 | yA |
| 15 | Chn | x44 | a14 | Chn | x34 | a13 | p14 | s9 | p13 | s8 | p10 | |

† Contents of this register are not valid during this cycle.

Chebyshev Routines for the SN74ACT8847 FPU

Introduction

Using the SN74ACT8847, very efficient routines can be developed for the implementation of transcendental functions. A high degree of accuracy can be achieved by taking advantage of the 'ACT8847's ability to perform calculations using double-precision floating point operands.

This application note describes how to use the 'ACT8847 to implement seven different transcendental functions. TIM (Texas Instruments Meta-Macro Assembler) assembly files have been written for all seven functions and these files are available upon request from Texas Instruments. The algorithm chosen to implement these functions is the Chebyshev expansion method [1]. Table 6 lists the functions that have been implemented, along with the number of cycles required, and time required to perform the calculations. Also listed in the table is the cycle count and time required to perform the same calculation using the Motorola MC68881 Floating Point Coprocessor and the Intel 80387 Numeric Processor Extension.

The Chebyshev expansion method was chosen rather than some of the more well known methods, such as the Taylor series and Newton-Raphson approximation, for a variety of reasons. The primary advantage of Chebyshev's method is that it provides a uniform convergence rate in the number of terms required to achieve the desired accuracy. Thus the range of the input value will have little effect on the accuracy of the result. Another advantage is that the number of terms required to calculate the

approximation is relatively small. This provides for faster execution. Also, Chebyshev's method can be applied to any function which is continuous and of bounded variation. Lastly, tables are available which contain the constants necessary to implement Chebyshev's method.

In order that this application note be useful to the largest audience, only those instructions and features common to all 'ACT8847 versions have been used to implement the routines.

Contact Texas Instruments VLSI Logic applications group at (214) 997-3970 for a copy of the seven TIM assembly files.

Table 6. Cycle Count and Execution Speed for the Seven Chebyshev Functions

| FUNCTION | CYCLE COUNT [†] | | | EXECUTION SPEED [‡] IN MICROSECONDS | | |
|----------------|--------------------------|---------|------------|---|---------|--------------|
| | 'ACT8847 | MC68881 | 80387 | 'ACT8847 | MC68881 | 80387 |
| Sine | 51 | 416 | 122 to 771 | 1.53 | 25.0 | 7.32 to 46.3 |
| Cosine | 51 | 416 | 123 to 772 | 1.53 | 25.0 | 7.38 to 46.3 |
| Tangent | 84 | 498 | 191 to 497 | 2.52 | 29.9 | 11.5 to 29.8 |
| ArcSine | 68 | 606 | Not Avail. | 2.04 | 36.4 | Not Avail. |
| ArcCosine | 68 | 650 | Not Avail. | 2.04 | 39.0 | Not Avail. |
| ArcTangent | 104 | 428 | 314 to 487 | 3.12 | 25.7 | 18.8 to 29.2 |
| Exponentiation | 52 | 522 | Not Avail. | 1.56 | 31.3 | Not Avail. |

[†]For MC68881 cycle count refer to 'MC68881 Floating Point Coprocessor User's Manual', Document No. MC68881UM/AD, Page 6-13. For 80387 cycle count refer to '80387 Programmer's Reference Manual', Document No. 231917-001, Page E-36.

[‡]'ACT8847 cycle speed is 30 ns, 33 MHz
MC68881 cycle speed is 60 ns, 16.6 MHz
80387 cycle speed is 40 ns, 25 MHz

Overview of Chebyshev's Expansion Method

If $f(x)$ is continuous and of bounded variation over the interval $-1 \leq x \leq 1$, then $f(x)$ may be approximated by the following equation:

$$\begin{aligned}
 f(x) &= 1/2a_0 + a_1T_1(x) + a_2T_2(x) + \dots \\
 &= \sum_{r=0}^{\infty} a_rT_r(x)
 \end{aligned}$$

Note that the range for x is between -1 and 1 . For most functions, this restriction requires that the input, x , be range reduced before the calculation begins. Range reducing an argument means to scale the argument down to a certain range. In the case of Chebyshev approximations, the range is usually $-1 \leq x \leq 1$, or $0 \leq x \leq 1$.

In the equation for $f(x)$ above, the constants represented by a_n are known as Chebyshev coefficients. The variables represented by T_r are known as Chebyshev polynomials and can be derived from the following relationship and values:

$$T_{r+1}(x) - 2xT_r(x) + T_{r-1}(x) = 0,$$

$$T_0(x) = 1,$$

$$T_1(x) = x$$

To illustrate Chebyshev's expansion method, the procedure to approximate function $f(x)$ using the first seven polynomials is now covered. Let

$$\begin{aligned} f(x) = & 1/2a_0 + \\ & a_1T_1(x) + \\ & a_2T_2(x) + \\ & a_3T_3(x) + \\ & a_4T_4(x) + \\ & a_5T_5(x) + \\ & a_6T_6(x) \end{aligned}$$

Substituting in the expressions for the polynomials,

$$\begin{aligned} f(x) = & 1/2a_0 + \\ & a_1(x) + \\ & a_2(2x^2 - 1) + \\ & a_3(4x^3 - 3x) + \\ & a_4(8x^4 - 8x^2 + 1) + \\ & a_5(16x^5 - 20x^3 + 5x) + \\ & a_6(32x^6 - 48x^4 + 18x^2 - 1) \end{aligned}$$

Rearranging the expression, by grouping powers of x ,

$$\begin{aligned} f(x) = & x^0(1/2a_0 - a_2 + a_4 - a_6) + \\ & x^1(a_1 - 3a_3 + 5a_5) + \\ & x^2(2a_2 - 8a_4 + 18a_6) + \\ & x^3(4a_3 - 20a_5) + \\ & x^4(8a_4 - 48a_6) + \\ & x^5(16a_5) + \\ & x^6(32a_6) \end{aligned}$$

Next make the following substitutions:

$$\text{Let } c_0 = 1/2a_0 - a_2 + a_4 - a_6$$

$$c_1 = a_1 - 3a_3 + 5a_5$$

$$c_2 = 2a_2 - 8a_4 + 18a_6$$

$$c_3 = 4a_3 - 20a_5$$

$$c_4 = 8a_4 - 48a_6$$

$$c_5 = 16a_5$$

$$c_6 = 32a_6$$

Substituting the c's into the last equation for f(x),

$$f(x) = c_0x^0 + c_1x^1 + c_2x^2 + c_3x^3 + c_4x^4 + c_5x^5 + c_6x^6$$

Applying Horner's Rule yields,

$$f(x) = (((((c_6x + c_5)x + c_4)x + c_3)x + c_2)x + c_1)x + c_0$$

In the remainder of the paper, the above equation will be referred to as C_{series} . Therefore,

$$C_{\text{series_f}}(x) = (((((c_6x + c_5)x + c_4)x + c_3)x + c_2)x + c_1)x + c_0$$

The last step prior to approximating f(x) is to calculate the c's by substituting the values for the Chebyshev coefficients into the equations for c_0 through c_6 .

Format for the Remainder of the Application Note

Each of the seven functions will be covered in a separate section. Each section will include the following information:

1. General steps required to perform the calculation including a description of any preprocessing and/or postprocessing
2. An algorithm for each of the above steps
3. What system intervention, if any, is required; this intervention may take the form of branching based on comparison status generated by the 'ACT8847, or storing and then later retrieving intermediate results
4. The number of 'ACT8847 cycles required to calculate f(x)
5. A listing of the c's
6. Pseudocode table showing how the calculation is accomplished. The pseudocode tables list the contents of all the relevant 'ACT8847 registers and buses for each instruction.
7. Microcode table listing the instructions

References

- [1] C. W. Clenshaw, G. F. Miller, and M. Woodger, "Algorithms for Special Functions I," Numerische Mathematik, Vol 4, 1963, pages 403 through 419.
- [2] C. W. Clenshaw, "Chebyshev Series for Mathematical Functions," Vol 5 of the Mathematical Tables of the National Physical Laboratory, Department of Scientific Industrial Research, England, 1960.

Cosine Routine Using Chebyshev's Method

All floating point inputs and outputs are double precision. The input is in radians.

Steps Required to Perform the Calculation

STEP 1 — Preprocessing; range-reduce the input, X , to a range of $[-1, 1]$. Next square this range-reduced value, multiply it by 2.0, and finally subtract 1.0. $X3$ is the range-reduced input value, it must be stored externally. 'TRUNC' means to truncate.

$$X1 \leftarrow X \cdot (2.0/\pi)$$

$$X2 \leftarrow (4(\text{TRUNC}(0.25(X1 + 2.0)))) - X1 + 1.0$$

If $X2 > 1.0$

$$\text{Then } X3 \leftarrow 2.0 - X2$$

$$\text{Else } X3 \leftarrow X2$$

$$X4 \leftarrow 2.0 \cdot (X3 \cdot X3) - 1.0$$

STEP 2 — Core Calculation; $X4$ in Step 1 will be referred to as ' x ' in the core calculation.

$$X5 \leftarrow C_{\text{series_cos}}$$

$$\leftarrow ((((((c8 \cdot x + c7) \cdot x + c6) \cdot x + c5) \cdot x + c4) \cdot x + c3) \cdot x + c2) \cdot x + c1) \cdot x + c0$$

STEP 3 — Postprocessing; multiply the output of the core calculation times $X3$.

$$\text{Cosine}(X) \leftarrow X5 \cdot X3$$

Algorithms for the Three Steps

Step 1 perform the preprocessing:

| | |
|--|--|
| T1 $\leftarrow X \cdot (2.0/\pi)$ | 2.0/ π entered as a constant |
| T2 $\leftarrow T1 + 2.0$ | |
| T3 $\leftarrow 0.25 \cdot T2$ and | CREG $\leftarrow T1, T3$ and $T4$ result |
| T4 $\leftarrow 1.0 - \text{CREG}$ | from a chained instruction |
| T5 $\leftarrow \text{INT}(T3)$ | round controls set to truncate |
| T6 $\leftarrow 4 \cdot T5$ | CREG $\leftarrow T4$ |
| T7 $\leftarrow \text{DOUBLE}(T6)$ | convert from integer to double |
| T8 $\leftarrow T7 + \text{CREG}$ | |
| CMP (1.0, T8) | |
| If (1.0 > T8) | CREG $\leftarrow T8$ |
| Then T9 $\leftarrow 2.0 - \text{CREG}$ | T9 is X3 in Step 1, must |
| Else T9 $\leftarrow \text{CREG}$ | be stored externally |
| | CREG $\rightarrow T9$ |
| T10 $\leftarrow \text{CREG} \cdot \text{CREG}$ | |
| T11 $\leftarrow T10 \cdot 2.0$ | |
| T12 $\leftarrow T11 - 1.0$ | T12 is X4 in Step 1, the |
| | input to the core routine |

Step 2 perform the core calculation:

| | |
|--|-----------------------|
| T13 $\leftarrow c_8 \cdot \text{CREG}$ | |
| T14 $\leftarrow T13 + c_7$ | CREG $\leftarrow T12$ |
| T15 $\leftarrow T14 \cdot \text{CREG}$ | |
| T16 $\leftarrow T15 + c_6$ | |
| T17 $\leftarrow T16 \cdot \text{CREG}$ | |
| T18 $\leftarrow T17 + c_5$ | |
| T19 $\leftarrow T18 \cdot \text{CREG}$ | |
| T20 $\leftarrow T19 + c_4$ | |
| T21 $\leftarrow T20 \cdot \text{CREG}$ | |
| T22 $\leftarrow T21 + c_3$ | |
| T23 $\leftarrow T22 \cdot \text{CREG}$ | |
| T24 $\leftarrow T23 + c_2$ | |
| T25 $\leftarrow T24 \cdot \text{CREG}$ | |
| T26 $\leftarrow T25 + c_1$ | |
| T27 $\leftarrow T26 \cdot \text{CREG}$ | |
| T28 $\leftarrow T27 + c_0$ | |

Step 3 perform the postprocessing:

Cosine(X) $\leftarrow T28 \cdot T9$

Required System Intervention

As seen in the algorithm for Step 1, the 'ACT8847 performs a compare. The results of this compare determine which one of two calculations is to be performed. The system, in which the 'ACT8847 is a part, must make the decision as to which of the two calculations is to be performed. In addition, the system must store X3 and then later furnish X3 as an input to the 'ACT8847.

Number of 'ACT8847 Cycles Required to Calculate Cosine(x)

Calculation of Cosine(x) requires 46 cycles. In addition, it is assumed that five additional cycles are required due to the compare instruction, and resulting system intervention. Therefore, the total number of cycles to perform the Cosine(x) calculation is 51.

Listing of the Chebyshev Constants (c's)

The constants are represented in IEEE double-precision floating point format.

| | | |
|----|---|------------------|
| c8 | = | 3D19D46B7D4C8F32 |
| c7 | = | BD962909C5C01ED6 |
| c6 | = | 3E0D53517735F927 |
| c5 | = | BE7CC930FD0ADA9D |
| c4 | = | 3EE3E0AF61F7677F |
| c3 | = | BF41E5FDEF25C403 |
| c2 | = | 3F92A9FB40C119ED |
| c1 | = | BFD23B03366AA0C9 |
| c0 | = | 3FF4464BCC8CBA1F |

Pseudocode Table for the Cosine(x) Calculation

Table 7. Pseudocode for Chebyshev Cosine Routine (PIPES2-0 = 010, RND1-0 = 00)

| CLK | DA BUS | DB BUS | RA REG | RB REG | CLK MODE | INSTR | MUL PIPE | ALU PIPE | P REG | C REG | S REG | Y BUS | COMMENT |
|-----|---------------|---------------|-----------|-------------|-------------|----------------------|-------------|-------------|----------|----------|----------|----------|---|
| 1 | X MSH | X LSH | | | 0 | RA2•RB2 | | | | | | | X is the input |
| 2 | 2DIVPI MSH | 2DIVPI LSH | X | 2DIVPI | 0 | RA2•RB2 | | | | | | | 2DIVPI is a constant representing 2.0/pi |
| 3 | 1.0 MSH | 1.0 LSH | X | 2DIVPI | 0 | PR4 + RB4 | RA2•RB2 | | | | | | Preload RA with 1.0 for use in cycles 5 and 11 |
| 4 | 2.0 MSH | 2.0 LSH | 1.0 | 2.0 | 0 | PR4 + RB4 | | | P1 | | | | |
| 5 | 0.25 MSH | 0.25 LSH | 1.0 | 0.25 | 1 | SR5•RB5 RA5 – CR5 | | | | P1 | S1 | | |
| 6 | | | 1.0 | 0.25 | 0 | DP2I(PR7) | SR5•RB5 | RA5 – CR5 | | | | | Double precision → integer |
| 7 | | | 1.0 | 0.25 | 0 | DP2I(PR7) | | | P2 | | S2 | | Cycles 6,7 set RND1, 0 = 01 |
| 8 | | 4 | 1.0 | 4 | 0 | SR8•RB8 | | | | S2 | S3 | | |
| 9 | | | 1.0 | 4 | 1 | I2DP(PR9) | | | P3 | | | | Integer → double-precision |
| 10 | | | 1.0 | 4 | 1 | CR10 + SR10 | | | | | S4 | | |
| 11 | | | 1.0 | 4 | 1 | COMPARE RA11,SR11 | | | | | S5 | | If SR11 > RA11 then 13a If SR11 ≤ RA11 then 13b |
| 12 | | | 1.0 | 4 | 0 | NOP | | | | S5 | | | Wait for system response |
| 13a | 2.0 MSH | 2.0 LSH | 1.0 | 2.0 | 1 | RB13 – CR13 | | | | | | | Execute 13a or 13b |
| 13b | | | 1.0 | 4 | 1 | PAS(CR13) | | | | | | | Pass contents of CREG |
| 14 | | | 1.0 | 2.0 or 4 | 1 | CR14•CR14 | | | | | S6 | S6 | S6 is either RB13-CR13 or CR13 from PASS CR13, and must be stored externally for use in cycle 43 |
| 15 | 2.0 MSH | 2.0 LSH | 1.0 | 2.0 or 4 | 0 | RA16•PR16 | CR14•CR14 | | | S6 | | S6 | Output S6 in cycles 14 and 15 |
| 16 | | | 2.0 | 2.0 or 4 | 0 | RA16•PR16 | | | P4 | | | | |

Table 7. Pseudocode for Chebyshev Cosine Routine (PIPES2-0 = 010, RND1-0 = 00) (Continued)

| CLK | DA BUS | DB BUS | RA REG | RB REG | CLK MODE | INSTR | MUL PIPE | ALU PIPE | P REG | C REG | S REG | Y BUS | COMMENT |
|-----|--------------------|--------------------|-----------|----------------|-------------|-----------|-------------|-------------|----------|----------|----------|----------|---------------------------|
| 17 | | | 2.0 | 2.0 or 4 | 0 | PR18+RB18 | RA16*PR16 | | | | | | |
| 18 | -1.0 MSH | -1.0 LSH | 2.0 | -1.0 | 0 | PR18+RB18 | | | P5 | | | | |
| 19 | c ₈ MSH | c ₈ LSH | 2.0 | c ₈ | 1 | SR19*RB19 | | | | | S7 | | Start core calculation |
| 20 | | | 2.0 | c ₈ | 0 | PR21+RB21 | SR19*RB19 | | | S7 | | | S7 is input to core calc. |
| 21 | c ₇ MSH | c ₇ LSH | 2.0 | c ₇ | 0 | PR21+RB21 | | | P6 | | | | |
| 22 | | | 2.0 | c ₇ | 1 | SR22*CR22 | | | | | S8 | | |
| 23 | | | 2.0 | c ₇ | 0 | PR24+RB24 | SR22*CR22 | | | | | | |
| 24 | c ₆ MSH | c ₆ LSH | 2.0 | c ₆ | 0 | PR24+RB24 | | | P7 | | | | |
| 25 | | | 2.0 | c ₆ | 1 | SR25*CR25 | | | | | S9 | | |
| 26 | | | 2.0 | c ₆ | 0 | PR27+RB27 | SR25*CR25 | | | | | | |
| 27 | c ₅ MSH | c ₅ LSH | 2.0 | c ₅ | 0 | PR27+RB27 | | | P8 | | | | |
| 28 | | | 2.0 | c ₅ | 1 | SR28*CR28 | | | | | S10 | | |
| 29 | | | 2.0 | c ₅ | 0 | PR30+RB30 | SR28*CR28 | | | | | | |
| 30 | c ₄ MSH | c ₄ LSH | 2.0 | c ₄ | 0 | PR30+RB30 | | | P9 | | | | |
| 31 | c ₃ MSH | c ₃ LSH | 2.0 | c ₄ | 1 | SR31*CR31 | | | | | S11 | | |
| 32 | c ₃ MSH | c ₃ LSH | 2.0 | c ₄ | 0 | PR33+RB33 | SR31*CR31 | | P10 | | | | |
| 33 | c ₃ MSH | c ₃ LSH | 2.0 | c ₃ | 0 | PR33+RB33 | | | | | S12 | | |
| 34 | | | 2.0 | c ₃ | 1 | SR34*CR34 | | | | | | | |
| 35 | c ₂ MSH | c ₂ LSH | 2.0 | c ₃ | 0 | PR36+RB36 | SR34*CR34 | | P11 | | | | |
| 36 | c ₂ MSH | c ₂ LSH | 2.0 | c ₂ | 0 | PR36+RB36 | | | | | | | |

Table 7. Pseudocode for Chebyshev Cosine Routine (PIPES2-0 = 010, RND1-0 = 00) (Continued)

Table 7. Pseudocode for Chebyshev Cosine Routine (PIPES2-0 = 010, RND1-0 = 00) (Concluded)

| CLK | DA BUS | DB BUS | RA REG | RB REG | CLK MODE | INSTR | MUL PIPE | ALU PIPE | P REG | C REG | S REG | Y BUS | COMMENT |
|-----|--------------------|--------------------|-----------|----------------|-------------|-----------|-------------|-------------|----------|----------|----------|----------|---|
| 37 | | | 2.0 | c ₂ | 1 | SR37*CR37 | | | | | S13 | | |
| 38 | c ₃ MSH | c ₃ LSH | 2.0 | c ₂ | 0 | PR39+RB39 | SR37*CR37 | | b11 | | | | |
| 39 | c ₁ MSH | c ₁ LSH | 2.0 | c ₁ | 0 | PR39+RB39 | SR37*CR37 | | P12 | | | | |
| 40 | | | 2.0 | c ₁ | 1 | SR40*CR40 | | | | | S14 | | |
| 41 | c ₃ MSH | c ₃ LSH | 2.0 | c ₁ | 0 | PR42+RB42 | SR40*CR40 | | b10 | | | | |
| 42 | c ₀ MSH | c ₀ LSH | 2.0 | c ₀ | 0 | PR42+RB42 | SR40*CR40 | | P13 | | | | |
| 43 | S6 MSH | S6 LSH | 2.0 | S6 | 1 | SR43*RB43 | S15 | | | | S11 | | Begin postprocessing |
| 44 | c ₄ MSH | c ₄ LSH | 2.0 | S6 | 0 | DUMMY | SR43*RB43 | | | | | | Instruction is double-precision RA + RB, allows time for answer to propagate to the Y bus |
| 45 | | | 2.0 | S6 | 0 | NOP | | | P14 | | | P14 | Output MSH of answer |
| 46 | | | 2.0 | S6 | 0 | NOP | | | P14 | | | P14 | Output LSH of answer |
| 32 | | | 2.0 | c ₄ | 0 | SR30*CR30 | | | b8 | | | | |
| 34 | c ₀ MSH | c ₀ LSH | 2.0 | c ₀ | 0 | SR31+RB31 | | | b3 | | | | |
| 33 | | | 2.0 | c ₃ | 0 | SR34+RB34 | SR35*CR33 | | | | | | |
| 35 | | | 2.0 | c ₁ | 1 | SR35*CR33 | | | | | S8 | | |
| 31 | c ₁ MSH | c ₁ LSH | 2.0 | c ₁ | 0 | SR31+RB31 | | | b9 | | | | |
| 30 | | | 2.0 | c ₀ | 0 | SR31+RB31 | SR18*RB18 | | | S1 | | | SR18 output is double-precision |
| 19 | c ₀ MSH | c ₀ LSH | 2.0 | c ₀ | 1 | SR18*RB18 | | | | | S1 | | SR18 output is double-precision |
| 18 | -1.0 MSH | -1.0 LSH | 2.0 | -1.0 | 0 | SR18+RB18 | | | b2 | | | | |
| 11 | | | 2.0 | c ₀ | 0 | SR18+RB18 | SR18*RB18 | | | | | | |
| CLK | DA BUS | DB BUS | RA REG | RB REG | CLK MODE | INSTR | MUL PIPE | ALU PIPE | P REG | C REG | S REG | Y BUS | COMMENT |

Table 7. Pseudocode for Chebyshev Cosine Routine (PIPES2-0 = 010, RND1-0 = 00) (Continued)

Microcode Table for the Cosine(x) Calculation

All numbers are in hex. Any field with a length that is not a multiple of 4 is right justified and zero filled. For the microcode table, the value of X has been chosen to be 1/2 pi.

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P | D | D | P | E | E | C | P | C | C | S | R | H | E | F | I | R | F | S | B | S | T | S | O | O | O |
| A | A | B | B | N | N | L | I | L | O | E | E | A | N | L | N | N | A | R | Y | E | E | E | E | E | E |
| | | | A | B | K | P | K | N | L | S | L | C | O | S | D | S | C | T | L | S | L | Y | S | C | |
| | | | | | C | E | M | F | O | E | T | | | W | T | | T | C | E | S | T | Y | | | |
| | | | | | S | O | I | P | T | | | | | C | R | | | | P | T | | | | | |
| | | | | | | D | G | | | | | | | | | | | | | | | | | | |

| | | | | | | |
|------------|----------|---------|-------|----------|-------|---------------------|
| F 3FF921FB | 54442D18 | F 0 0 _ | 2 0 3 | FF 1 1 1 | 0 1C0 | 0 0 0 0 3 3 1 0 0 0 |
| F 3FE45F30 | 6DC9C883 | F 1 1 _ | 2 0 3 | FF 1 1 1 | 0 1C0 | 0 0 0 0 3 3 1 0 0 0 |
| F 3FF00000 | 00000000 | F 0 0 _ | 2 0 3 | FB 1 1 1 | 0 180 | 0 0 0 0 3 3 1 0 0 0 |
| F 40000000 | 00000000 | F 1 1 _ | 2 0 3 | FB 1 1 1 | 0 180 | 0 0 0 0 3 3 1 0 0 0 |
| F 3FD00000 | 00000000 | F 0 1 J | 2 1 3 | BD 1 1 0 | 0 581 | 0 0 1 0 3 3 1 0 0 0 |
| F 00000000 | 00000000 | F 0 0 _ | 2 0 3 | FB 1 1 1 | 0 1A3 | 1 0 0 0 3 3 1 0 0 0 |
| F 00000000 | 00000000 | F 0 0 _ | 2 0 3 | FB 1 1 1 | 0 1A3 | 1 0 0 0 3 3 1 0 0 0 |
| F 00000000 | 00000004 | F 0 1 J | 2 0 1 | BF 1 1 0 | 0 240 | 0 0 0 0 3 3 1 0 0 0 |
| F 00000000 | 00000000 | F 0 0 _ | 2 1 3 | FB 1 1 1 | 0 1A2 | 0 0 0 0 3 3 1 0 0 0 |
| F 00000000 | 00000000 | F 0 0 _ | 2 1 3 | F6 1 1 1 | 0 180 | 0 0 0 0 3 3 1 0 0 0 |
| F 00000000 | 00000000 | F 0 0 _ | 2 1 3 | FE 1 1 1 | 0 182 | 0 0 0 0 3 3 1 0 0 0 |
| F 00000000 | 00000000 | F 0 0 J | 2 0 3 | FF 1 1 0 | 0 300 | 0 0 0 0 3 3 1 0 0 0 |
| F 00000000 | 00000000 | F 0 0 _ | 2 1 3 | F7 1 1 1 | 0 1A0 | 0 0 0 0 3 3 1 0 0 0 |
| F 00000000 | 00000000 | F 0 0 _ | 2 1 3 | 5F 1 1 1 | 0 1C0 | 0 0 0 0 3 3 1 0 0 0 |
| F 40000000 | 00000000 | F 0 0 J | 2 0 3 | EF 1 1 0 | 0 1C0 | 0 0 0 0 3 3 1 0 0 0 |
| F 00000000 | 00000000 | F 1 0 _ | 2 0 3 | EF 1 1 1 | 0 1C0 | 0 0 0 0 3 3 1 0 0 0 |
| F 00000000 | 00000000 | F 0 0 _ | 2 0 3 | FB 1 1 1 | 0 180 | 0 0 0 0 3 3 1 0 0 0 |
| F BFF00000 | 00000000 | F 0 1 _ | 2 0 3 | FB 1 1 1 | 0 180 | 0 0 0 0 3 3 1 0 0 0 |
| F 3D19D46B | 7D4C8F32 | F 0 1 _ | 2 1 3 | BF 1 1 1 | 0 1C0 | 0 0 0 0 3 3 1 0 0 0 |
| F 00000000 | 00000000 | F 0 0 J | 2 0 3 | FB 1 1 0 | 0 180 | 0 0 0 0 3 3 1 0 0 0 |
| F BD962909 | C5C01ED6 | F 0 1 _ | 2 0 3 | FB 1 1 1 | 0 180 | 0 0 0 0 3 3 1 0 0 0 |

Microcode Table for the Cosine(x) Calculation (Continued)

| P | D | D | P | E | E | C | P | C | C | S | R | H | E | F | I | R | F | S | B | S | T | S | O | O | O |
|------------|----------|-------|---|---|---|---|----|---|---|---|---|-----|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | A | B | B | N | N | L | I | L | O | E | E | A | N | L | N | N | A | R | Y | E | E | E | E | E | E |
| | | | A | B | K | P | K | N | L | S | L | C | O | S | D | S | C | T | L | S | L | Y | S | C | |
| | | | | | C | E | M | F | O | E | T | W | T | | | T | C | E | S | T | Y | | | | |
| | | | | | | S | O | I | P | T | | | C | R | | | P | T | | | | | | | |
| | | | | | | D | G | | | | | | | | | | | | | | | | | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 3E0D5351 | 7735F927 | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F BE7CC930 | FD0ADA9D | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 3EE3E0AF | 61F7677F | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F BF41E5FD | EF25C403 | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 3F92A9FB | 40C119ED | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F BFD23B03 | 366AA0C9 | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 3FF4464B | CC8CBA1F | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 0 1 | — | 2 | 1 | 3 | BF | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FF | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FF | 1 | 1 | 1 | 0 | 300 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FF | 1 | 1 | 1 | 0 | 300 | 0 | 0 | 0 | 0 | 3 | 3 | 0 | 0 | 0 | 0 | | | |

Sine Routine Using Chebyshev's Method

All floating point inputs and outputs are double precision. The input is in radians.

Steps Required to Perform the Calculation

STEP 1 — Preprocessing; range reduce the input, X, to a range of $[-1, 1]$. Next square this range-reduced value, multiply it by 2.0, and finally subtract 1.0. X3 is the range-reduced input value, it must be stored externally. 'TRUNC' means to truncate.

```
X1 ← X*(2.0/pi)
X2 ← X1 - (4*(TRUNC(0.25*(X1 + 1.0))))
If X2 > 1.0
    Then X3 ← 2.0 - X2
    Else X3 ← X2
X4 ← 2.0*(X3*X3) - 1.0
```

STEP 2 — Core calculation; X4 in Step 1 will be referred to as 'x' in the core calculation.

```
X5 ← Cseries_sin
← ((((((c8**x + c7)*x + c6)*x + c5)*x +
c4)*x + c3)*x + c2)*x + c1)*x + c0
```

STEP 3 — Postprocessing; multiply the output of the core calculation times X3.

Sine(X) ← X5*X3

Algorithms for the Three Steps

Step 1 perform the preprocessing:

| | |
|----------------------|---|
| T1 ← X*(2.0/pi) | 2.0/pi entered as a constant |
| T2 ← T1 + 1.0 | |
| T3 ← 0.25*T2 | CREG ← T1 |
| T4 ← INT(T3) | round controls set to truncate |
| T5 ← 4*T4 | |
| T6 ← DOUBLE(T5) | convert from integer to double |
| T7 ← CREG - T6 | |
| CMP (1.0, T7) | compare 1.0 to T7 |
| If (1.0 > T7) | CREG ← T7 |
| Then T8 ← 2.0 - CREG | T8 is X3 in Step 1, must |
| Else T8 ← CREG | be stored externally |
| | CREG → T8 |
| T9 ← CREG*CREG | |
| T10 ← T9 * 2.0 | |
| T11 ← T10 - 1.0 | T11 is X4 in Step 1 above, the input to |
| | the core routine |
| | T11 = 'x' from Step 2 above |

Step 2 perform the core calculation:

```

T12 ← c8 * CREG      CREG ← T11
T13 ← T12 + c7
T14 ← T13 * CREG
T15 ← T14 + c6
T16 ← T15 * CREG
T17 ← T16 + c5
T18 ← T17 * CREG
T19 ← T18 + c4
T20 ← T19 * CREG
T21 ← T20 + c3
T22 ← T21 * CREG
T23 ← T22 + c2
T24 ← T23 * CREG
T25 ← T24 + c1
T26 ← T25 * CREG
T27 ← T26 + c0

```

Step 3 perform the postprocessing:

Sine(X) ← T27 * T8

Required System Intervention

As seen in the algorithm for Step 1, the 'ACT8847 performs a compare. The results of this compare determine which one of two calculations is to be performed. The system, in which the 'ACT8847 is a part, must make the decision between which two calculations are to be performed. In addition, the system must store X3 and then later furnish X3 as an input to the 'ACT8847.

Number of 'ACT8847 Cycles Required to Calculate Sine(x)

Calculation of Sine(x) requires 46 cycles. In addition, it is assumed that five additional cycles are required due to the compare instruction and resulting system intervention. Therefore, the total number of cycles to perform the Sine(x) calculation is 51.

Listing of the Chebyshev Constants (c's)

The constants are represented in IEEE double-precision floating point format.

```

c8 = 3D19D46B7D4C8F32
c7 = BD962909C5C01ED6
c6 = 3E0D53517735F927
c5 = BE7CC930FD0ADA9D
c4 = 3EE3E0AF61F7677F
c3 = BF41E5FDEF25C403
c2 = 3F92A9FB40C119ED
c1 = BFD23B03366AA0C9
c0 = 3FF4464BCC8CBA1F

```

Pseudocode Table for the Sine(x) Calculation

Table 8. Pseudocode for Chebyshev Sine Routine (PIPES2-0 = 010, RND1-0 = 00)

| CLK | DA BUS | DB BUS | RA REG | RB REG | CLK MODE | INSTR | MUL PIPE | ALU PIPE | P REG | C REG | S REG | Y BUS | COMMENT |
|-----|---------------|---------------|-----------|-------------|-------------|----------------------|-------------|-------------|----------|----------|----------|----------|---|
| 1 | X MSH | X LSH | | | 0 | RA2*RB2 | | | | | | | X is the input |
| 2 | 2DIVPI MSH | 2DIVPI LSH | X | 2DIVPI | 0 | RA2*RB2 | | | | | | | 2DIVPI is a constant representing 2.0/pi |
| 3 | | | X | 2DIVPI | 0 | PR4 + RB4 | RA2*RB2 | | | | | | |
| 4 | 1.0 MSH | 1.0 LSH | X | 1.0 | 0 | PR4 + RB4 | | | P1 | | | | |
| 5 | 0.25 MSH | 0.25 LSH | X | 0.25 | 1 | SR5*RB5 | | | | P1 | S1 | | |
| 6 | 1.0 MSH | 1.0 LSH | X | 0.25 | 0 | DP2I(PR7) | SR5*RB5 | | | | | | Double precision → integer |
| 7 | | | 1.0 | 0.25 | 0 | DP2I(PR7) | | | P2 | | | | Cycles 6,7 set RND1,0 = 01 |
| 8 | | 4 | 1.0 | 4 | 0 | SR8*RB8 | | | | | S2 | | |
| 9 | | | 1.0 | 4 | 1 | I2DP(PR9) | | | P3 | | | | Integer → double precision |
| 10 | | | 1.0 | 4 | 1 | CR10 – SR10 | | | | | S3 | | |
| 11 | | | 1.0 | 4 | 1 | COMPARE RA11,SR11 | | | | | S4 | | If SR11 → RA11 then 13a If SR11 ≤ RA11 then 13b |
| 12 | | | 1.0 | 4 | 0 | NOP | | | | S4 | | | Wait for system response |
| 13a | 2.0 MSH | 2.0 LSH | 1.0 | 2.0 | 1 | RB13 – CR13 | | | | | | | Execute 13a or 13b |
| 13b | | | 1.0 | 4 | 1 | PAS(CR13) | | | | | | | Pass contents of CREG |
| 14 | | | 1.0 | 2.0 or 4 | 1 | CR14*CR14 | | | | | S5 | S5 | S5 is either RB13 – CR13 or CR13 from PASS CR13, and must be stored externally for use in cycle 43 |
| 15 | 2.0 MSH | 2.0 LSH | 1.0 | 2.0 or 4 | 0 | RA16*PR16 | CR14*CR14 | | | S5 | | S5 | Output S5 in cycles 14 and 15 |
| 16 | | | 2.0 | 2.0 or 4 | 0 | RA16*PR16 | | | P4 | | | | |
| 17 | | | 2.0 | 2.0 or 4 | 0 | PR18 + RB18 | RA16*PR16 | | | | | | |
| 18 | -1.0 MSH | -1.0 LSH | 2.0 | -1.0 | 0 | PR18 + RB18 | | | P5 | | | | |

Table 8. Pseudocode for Chebyshev Sine Routine (PIPES2-0 = 010, RND1-0 = 00) (Continued)

| CLK | DA BUS | DB BUS | RA REG | RB REG | CLK MODE | INSTR | MUL PIPE | ALU PIPE | P REG | C REG | S REG | Y BUS | COMMENT |
|-----|--------------------|--------------------|-----------|----------------|-------------|-----------|-------------|-------------|----------|----------|----------|----------|---------------------------|
| 19 | c ₈ MSH | c ₈ LSH | 2.0 | c ₈ | 1 | SR19+RB19 | | | | | S6 | | Start core calculation |
| 20 | | | 2.0 | c ₈ | 0 | PR21+RB21 | SR19+RB19 | | | S6 | | | S7 is input to core calc. |
| 21 | c ₇ MSH | c ₇ LSH | 2.0 | c ₇ | 0 | PR21+RB21 | | | P6 | | | | |
| 22 | | | 2.0 | c ₇ | 1 | SR22+CR22 | | | | | S7 | | |
| 23 | | | 2.0 | c ₇ | 0 | PR24+RB24 | SR22+CR22 | | | | | | |
| 24 | c ₆ MSH | c ₆ LSH | 2.0 | c ₆ | 0 | PR24+RB24 | | | P7 | | | | |
| 25 | | | 2.0 | c ₆ | 1 | SR25+CR25 | | | | | S8 | | |
| 26 | | | 2.0 | c ₆ | 0 | PR27+RB27 | SR25+CR25 | | | | | | |
| 27 | c ₅ MSH | c ₅ LSH | 2.0 | c ₅ | 0 | PR27+RB27 | | | P8 | | | | |
| 28 | | | 2.0 | c ₅ | 1 | SR28+CR28 | | | | | S9 | | |
| 29 | | | 2.0 | c ₅ | 0 | PR30+RB30 | SR28+CR28 | | | | | | |
| 30 | c ₄ MSH | c ₄ LSH | 2.0 | c ₄ | 0 | PR30+RB30 | | | P9 | | | | |
| 31 | | | 2.0 | c ₄ | 1 | SR31+CR31 | | | | | S10 | | |
| 32 | | | 2.0 | c ₄ | 0 | PR33+RB33 | SR31+CR31 | | | | | | |
| 33 | c ₃ MSH | c ₃ LSH | 2.0 | c ₃ | 0 | PR33+RB33 | | | P10 | | | | |
| 34 | | | 2.0 | c ₃ | 1 | SR34+CR34 | | | | | S11 | | |
| 35 | | | 2.0 | c ₃ | 0 | PR36+RB36 | SR34+CR34 | | | | | | |
| 36 | c ₂ MSH | c ₂ LSH | 2.0 | c ₂ | 0 | PR36+RB36 | | | P11 | | | | |
| 37 | | | 2.0 | c ₂ | 1 | SR37+CR37 | | | | | S12 | | |
| 38 | | | 2.0 | c ₂ | 0 | PR39+RB39 | SR37+CR37 | | | | | | |
| 39 | c ₁ MSH | c ₁ LSH | 2.0 | c ₁ | 0 | PR39+RB39 | | | P12 | | | | |
| 40 | | | 2.0 | c ₁ | 1 | SR40+CR40 | | | | | S13 | | |

Pseudocode Table for the Sine(x) Calculation

Table 8. Pseudocode for Chebyshev Sine Routine (PIPES2-0 = 010, RND1-0 = 00) (Concluded)

| CLK | DA BUS | DB BUS | RA REG | RB REG | CLK MODE | INSTR | MUL PIPE | ALU PIPE | P REG | C REG | S REG | Y BUS | COMMENT |
|-----|--------------------|--------------------|-----------|----------------|-------------|-------------|-------------|-------------|----------|----------|----------|----------|---|
| 41 | | | 2.0 | c ₁ | 0 | PR42 + RB42 | SR40 * CR40 | | | | | | |
| 42 | c ₀ MSH | c ₀ LSH | 2.0 | c ₀ | 0 | PR42 + RB42 | | | P13 | | | | |
| 43 | S5 MSH | S5 LSH | 2.0 | S5 | 1 | SR43 * RB43 | | | | | S14 | | Begin postprocessing |
| 44 | | | 2.0 | S5 | 0 | DUMMY | SR43 * RB43 | | | | | | Instruction is double-precision RA + RB, allows time for answer to propagate to the Y bus |
| 45 | | | 2.0 | S5 | 0 | NOP | | | P14 | | | P14 | Output MSH of answer |
| 46 | | | 2.0 | S5 | 0 | NOP | | | P14 | | | P14 | Output LSH of answer |

Microcode Table for the Sine(x) Calculation

All numbers are in hex. Any field with a length that is not a multiple of 4 is right justified and zero filled. For the microcode table, the value of X has been chosen to be $1/2 \pi$.

| P | D | D | P | E | E | C | P | C | C | S | \bar{R} | \bar{H} | E | F | I | R | F | S | B | S | T | S | \bar{O} | \bar{O} | \bar{O} |
|------------|----------|-------|---|---|---|---|----|---|---|---|-----------|-----------|---|---|---|---|---|---|---|---|---|---|-----------|-----------|-----------|
| A | A | B | B | N | N | L | I | L | O | E | E | A | N | L | N | N | A | R | Y | E | E | E | E | E | E |
| | | | | A | B | K | P | K | N | L | S | L | C | O | S | D | S | C | T | L | S | L | Y | S | C |
| | | | | | | C | E | M | F | O | E | T | | W | T | | T | C | E | S | T | Y | | | |
| | | | | | | | S | O | I | P | T | | | C | R | | | | P | T | | | | | |
| | | | | | | D | G | | | | | | | | | | | | | | | | | | |
| F 3FF921FB | 54442D18 | F 0 0 | — | 2 | 0 | 3 | FF | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 3FE45F30 | 6DC9C883 | F 1 1 | — | 2 | 0 | 3 | FF | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 3FF00000 | 00000000 | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 3FD00000 | 00000000 | F 0 1 | ┐ | 2 | 1 | 3 | BF | 1 | 1 | 0 | 0 | 1C0 | 0 | 0 | 1 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 3FF00000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 1A3 | 1 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 1 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 1A3 | 1 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000004 | F 0 1 | — | 2 | 0 | 1 | BF | 1 | 1 | 1 | 0 | 240 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | FB | 1 | 1 | 1 | 0 | 1A2 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | F6 | 1 | 1 | 1 | 0 | 181 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | FE | 1 | 1 | 1 | 0 | 182 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 0 0 | ┐ | 2 | 0 | 3 | FF | 1 | 1 | 0 | 0 | 300 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | F7 | 1 | 1 | 1 | 0 | 1A0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 5F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 40000000 | 00000000 | F 0 0 | ┐ | 2 | 0 | 3 | EF | 1 | 1 | 0 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 1 0 | — | 2 | 0 | 3 | EF | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F BFF00000 | 00000000 | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 3D19D46B | 7D4C8F32 | F 0 1 | — | 2 | 1 | 3 | BF | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F 00000000 | 00000000 | F 0 0 | ┐ | 2 | 0 | 3 | FB | 1 | 1 | 0 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |
| F BD962909 | C5C01ED6 | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | |

Microcode Table for the Sine(x) Calculation (Continued)

| P A | D A | D B | P B | E N | E N | C L | P P | C K | C N | S L | R S | H L | E C | F O | I S | R D | F S | S C | B T | S L | T S | S Y | 0 Y | 0 S | 0 C |
|------------|----------|---------|--------|--------|--------|--------|-------------|---------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| | | | | A | B | K | P | K | N | L | S | L | C | O | S | D | S | C | T | L | S | L | Y | S | C |
| | | | | | | | E | M | F | O | E | T | W | T | C | | | | | | | | | | |
| | | | | | | | S | O | I | P | T | | | | | | | | | | | | | | |
| | | | | | | | D | G | | | | | | | | | | | | | | | | | |
| F 00000000 | 00000000 | F 0 0 — | 2 1 3 | 9F | 1 1 1 | 0 1C0 | 0 0 0 0 3 3 | 1 0 0 0 | | | | | | | | | | | | | | | | | |
| F 00000000 | 00000000 | F 0 0 — | 2 0 3 | FB | 1 1 1 | 0 180 | 0 0 0 0 3 3 | 1 0 0 0 | | | | | | | | | | | | | | | | | |
| F 3E0D5351 | 7735F927 | F 0 1 — | 2 0 3 | FB | 1 1 1 | 0 180 | 0 0 0 0 3 3 | 1 0 0 0 | | | | | | | | | | | | | | | | | |
| F 00000000 | 00000000 | F 0 0 — | 2 1 3 | 9F | 1 1 1 | 0 1C0 | 0 0 0 0 3 3 | 1 0 0 0 | | | | | | | | | | | | | | | | | |
| F 00000000 | 00000000 | F 0 0 — | 2 0 3 | FB | 1 1 1 | 0 180 | 0 0 0 0 3 3 | 1 0 0 0 | | | | | | | | | | | | | | | | | |
| F BE7CC930 | FD0ADA9D | F 0 1 — | 2 0 3 | FB | 1 1 1 | 0 180 | 0 0 0 0 3 3 | 1 0 0 0 | | | | | | | | | | | | | | | | | |
| F 00000000 | 00000000 | F 0 0 — | 2 1 3 | 9F | 1 1 1 | 0 1C0 | 0 0 0 0 3 3 | 1 0 0 0 | | | | | | | | | | | | | | | | | |
| F 00000000 | 00000000 | F 0 0 — | 2 0 3 | FB | 1 1 1 | 0 180 | 0 0 0 0 3 3 | 1 0 0 0 | | | | | | | | | | | | | | | | | |
| F 3EE3E0AF | 61F7677F | F 0 1 — | 2 0 3 | FB | 1 1 1 | 0 180 | 0 0 0 0 3 3 | 1 0 0 0 | | | | | | | | | | | | | | | | | |
| F 00000000 | 00000000 | F 0 0 — | 2 1 3 | 9F | 1 1 1 | 0 1C0 | 0 0 0 0 3 3 | 1 0 0 0 | | | | | | | | | | | | | | | | | |
| F 00000000 | 00000000 | F 0 0 — | 2 0 3 | FB | 1 1 1 | 0 180 | 0 0 0 0 3 3 | 1 0 0 0 | | | | | | | | | | | | | | | | | |
| F BF41E5FD | EF25C403 | F 0 1 — | 2 0 3 | FB | 1 1 1 | 0 180 | 0 0 0 0 3 3 | 1 0 0 0 | | | | | | | | | | | | | | | | | |
| F 00000000 | 00000000 | F 0 0 — | 2 1 3 | 9F | 1 1 1 | 0 1C0 | 0 0 0 0 3 3 | 1 0 0 0 | | | | | | | | | | | | | | | | | |
| F 00000000 | 00000000 | F 0 0 — | 2 0 3 | FB | 1 1 1 | 0 180 | 0 0 0 0 3 3 | 1 0 0 0 | | | | | | | | | | | | | | | | | |
| F 3F92A9FB | 40C119ED | F 0 1 — | 2 0 3 | FB | 1 1 1 | 0 180 | 0 0 0 0 3 3 | 1 0 0 0 | | | | | | | | | | | | | | | | | |
| F 00000000 | 00000000 | F 0 0 — | 2 1 3 | 9F | 1 1 1 | 0 1C0 | 0 0 0 0 3 3 | 1 0 0 0 | | | | | | | | | | | | | | | | | |
| F 00000000 | 00000000 | F 0 0 — | 2 0 3 | FB | 1 1 1 | 0 180 | 0 0 0 0 3 3 | 1 0 0 0 | | | | | | | | | | | | | | | | | |
| F BFD23B03 | 366AA0C9 | F 0 1 — | 2 0 3 | FB | 1 1 1 | 0 180 | 0 0 0 0 3 3 | 1 0 0 0 | | | | | | | | | | | | | | | | | |
| F 00000000 | 00000000 | F 0 0 — | 2 1 3 | 9F | 1 1 1 | 0 1C0 | 0 0 0 0 3 3 | 1 0 0 0 | | | | | | | | | | | | | | | | | |
| F 00000000 | 00000000 | F 0 0 — | 2 0 3 | FB | 1 1 1 | 0 180 | 0 0 0 0 3 3 | 1 0 0 0 | | | | | | | | | | | | | | | | | |
| F 3FF4464B | CC8CBA1F | F 0 1 — | 2 0 3 | FB | 1 1 1 | 0 180 | 0 0 0 0 3 3 | 1 0 0 0 | | | | | | | | | | | | | | | | | |
| F 3FF00000 | 00000000 | F 0 1 — | 2 1 3 | BF | 1 1 1 | 0 1C0 | 0 0 0 0 3 3 | 1 0 0 0 | | | | | | | | | | | | | | | | | |
| F 00000000 | 00000000 | F 0 0 — | 2 0 3 | FF | 1 1 1 | 0 180 | 0 0 0 0 3 3 | 1 0 0 0 | | | | | | | | | | | | | | | | | |
| F 00000000 | 00000000 | F 0 0 — | 2 0 3 | FF | 1 1 1 | 0 300 | 0 0 0 0 3 3 | 1 0 0 0 | | | | | | | | | | | | | | | | | |
| F 00000000 | 00000000 | F 0 0 — | 2 0 3 | FF | 1 1 1 | 0 300 | 0 0 0 0 3 3 | 0 0 0 0 | | | | | | | | | | | | | | | | | |

Tangent Routine Using Chebyshev's Method

All floating-point inputs and outputs are double precision. The input is in radians.

Steps Required to Perform the Calculation

STEP 1 — Preprocessing; range reduce the input, X, to a range of $[-1, 1]$. Next square this range-reduced value, multiply it by 2.0, and finally subtract 1.0. X3 is the range-reduced input value, it must be stored externally. 'TRUNC' means to truncate. If $X2 > 1.0$, then in the postprocessing part of the routine, the answer is the reciprocal of $X5 \cdot X3$.

```
X1 ← X*(4.0/pi)
X2 ← X1 - (4*(TRUNC(0.25(X1 + 1.0))))
If X2 > 1.0
    Then X3 ← 2.0 - X2
    Else X3 ← X2
X4 ← 2.0*(X3*X3) - 1.0
```

STEP 2 — Core Calculation; X4 in Step 1 will be referred to as 'x' in the core calculation.

```
X5 ← Cseries_tan
← ((((((((((c14)*x + c13)*x + c12)*x + c11)*x + c10)*x +
c9)*x + c8)*x + c7)*x + c6)*x + c5)*x + c4)*x + c3)*x +
c2)*x + c1)*x + c0
```

STEP 3 — Postprocessing; multiply the output of the core calculation times X3. If $X2 > 1.0$, then the reciprocal of $X5 \cdot X3$ is the answer, if $X2 \leq 1.0$ then $X5 \cdot X3$ is the answer.

Tangent(X) ← $X5 \cdot X3$ (or reciprocal of $X5 \cdot X3$)

Algorithms for the Three Steps

Step 1 perform the preprocessing:

| | |
|----------------------|--------------------------------|
| T1 ← X*(4.0/pi) | 4.0/pi entered as a constant |
| T2 ← T1 + 1.0 | |
| T3 ← 0.25*T2 | CREG ← T1 |
| T4 ← INT(T3) | round controls set to truncate |
| T5 ← 4*T4 | |
| T6 ← DOUBLE(T5) | convert from integer to double |
| T7 ← CREG - T6 | |
| CMP (1.0, T7) | |
| If (1.0 > T7) | CREG ← T7 |
| Then T8 ← 2.0 - CREG | T8 is X3 in Step 1, must |
| Else T8 ← CREG | be stored externally |

T9 ← CREG*CREG

CREG ← T8

T10 ← T9*2.0

T11 ← T10 - 1.0

T11 is X4 in Step 1, the
input to the core routine

Step 2 perform the core calculation:

T12 ← c14*CREG

CREG ← T11

T13 ← T12 + c13

T14 ← T13*CREG

T15 ← T14 + c12

T16 ← T15*CREG

T17 ← T16 + c11

T18 ← T17*CREG

T19 ← T18 + c10

T20 ← T19*CREG

T21 ← T20 + c9

T22 ← T21*CREG

T23 ← T22 + c8

T24 ← T23*CREG

T25 ← T24 + c7

T26 ← T25*CREG

T27 ← T26 + c6

T28 ← T27*CREG

T29 ← T28 + c5

T30 ← T29*CREG

T31 ← T30 + c4

T32 ← T31*CREG

T33 ← T32 + c3

T34 ← T33*CREG

T35 ← T34 + c2

T36 ← T35*CREG

T37 ← T36 + c1

T38 ← T37*CREG

T39 ← T38 + c0

Step 3 perform the postprocessing:

T40 ← T39*T8

If X2 (in Step 1) > 1.0

Then Tangent(X) ← 1.0/T40

Else Tangent(X) ← T40

Required System Intervention

As seen in the algorithm for Step 1, the 'ACT8847 performs a compare. The results of this compare determine which one of two calculations is to be performed. The system, in which the 'ACT8847 is a part, must make the decision as to which of the two calculations is to be performed. In addition, the system must store X3 and then later furnish X3 as an input to the 'ACT8847. Finally, the system will have to determine if it is necessary to take the reciprocal of the final product (T40 in the Algorithm for Step 3) to yield the answer. If it is necessary to take the reciprocal, then the system will be required to direct the variable T40 from the 'ACT8847's output bus to the input buses. This is because operands for division instructions must be provided by the RA and RB registers; feedback is not an option.

Number of 'ACT8847 Cycles Required to Calculate Tangent(x)

Calculation of Tangent(x) requires 79 cycles. In addition, it is assumed that five additional cycles are required for system intervention due to the compare instruction. Therefore, the total number of cycles required to perform the Tangent(x) calculation is 84.

Listing of the Chebyshev Constants (c's)

The constants are represented in IEEE double-precision floating-point format.

c14 = 3D747D842210CC35
c13 = 3DA1D66636043991
c12 = 3DCCD078F52B3A73
c11 = 3DF938F9CDDFF864
c10 = 3E2620430E99B5B7
c9 = 3E535C2C953CE515
c8 = 3E80F07AFC099D7F
c7 = 3EADA4D789EB45C4
c6 = 3ED9F03D4C51A771
c5 = 3F06B236DE4D014C
c4 = 3F33DBFB01B3F415
c3 = 3F6160DE701F3A53
c2 = 3F8E70A18736FC10
c1 = 3FBAEA2653199611
c0 = 3FEC14B2675B10BA

Pseudocode Table for the Tangent(x) Calculation

Table 9. Pseudocode for Chebyshev Tangent Routine (PIPES2-0 = 010, RND1-0 = 00)

| CLK | DA BUS | DB BUS | RA REG | RB REG | CLK MODE | INSTR | MUL PIPE | ALU PIPE | P REG | C REG | S REG | Y BUS | COMMENT |
|-----|---------------|---------------|-----------|-------------|-------------|----------------------|-------------|-------------|----------|----------|----------|----------|---|
| 1 | X MSH | X LSH | | | 0 | RA2*RB2 | | | | | | | X is the input |
| 2 | 4DIVPI MSH | 4DIVPI LSH | X | 4DIV PI | 0 | RA2*RB2 | | | | | | | 4DIVPI is a constant representing 4.0/pi |
| 3 | | | X | 4DIVPI | 0 | PR4 + RB4 | RA2*RB2 | | | | | | |
| 4 | 1.0 MSH | 1.0 LSH | X | 1.0 | 0 | PR4 + RB4 | | | P1 | | | | |
| 5 | 0.25 MSH | 0.25 LSH | X | 0.25 | 1 | SR5*RB5 | | | | P1 | S1 | | |
| 6 | 1.0 MSH | 1.0 LSH | X | 0.25 | 0 | DP2I(PR7) | SR5*RB5 | | | | | | Double precision → integer |
| 7 | | | 1.0 | 0.25 | 0 | DP2I(PR7) | | | P2 | | | | Cycles 6,7 set RND1,0 = 01 |
| 8 | | 4 | 1.0 | 4 | 0 | SR8*RB8 | | | | | S2 | | |
| 9 | | | 1.0 | 4 | 1 | I2DP(PR9) | | | P3 | | | | Integer → double precision |
| 10 | | | 1.0 | 4 | 1 | CR10-SR10 | | | | | S3 | | |
| 11 | | | 1.0 | 4 | 1 | COMPARE RA11,SR11 | | | | | S4 | | If SR11 > RA11 then 13a If SR11 ≤ RA11 then 13b |
| 12 | | | 1.0 | 4 | 0 | NOP | | | | S4 | | | Wait for system response |
| 13a | 2.0 MSH | 2.0 LSH | 1.0 | 2.0 | 1 | RB13 - CR13 | | | | | | | Execute 13a or 13b |
| 13b | | | 1.0 | 4 | 1 | PAS(CR13) | | | | | | | Pass contents of Creg |
| 14 | | | 1.0 | 2.0 or 4 | 1 | CR14*CR14 | | | | | S5 | S5 | S5 is either RB13-CR13 or CR13 from PASS CR13, and must be stored externally for use in cycle 61 |
| 15 | 2.0 MSH | 2.0 LSH | 1.0 | 2.0 or 4 | 0 | RA16*PR16 | CR14*CR14 | | | S5 | | S5 | Output S5 in cycles 14 and 15 |
| 16 | | | 2.0 | 2.0 or 4 | 0 | RA16*PR16 | | | P4 | | | | |
| 17 | | | 2.0 | 2.0 or 4 | 0 | PR18 + RB18 | RA16*PR16 | | | | | | |
| 18 | -1.0 MSH | -1.0 LSH | 2.0 | -1.0 | 0 | PR18 + RB18 | | | P5 | | | | |

Table 9. Pseudocode for Chebyshev Tangent Routine (PIPES2-0 = 010, RND1-0 = 00) (Continued)

| CLK | DA BUS | DB BUS | RA REG | RB REG | CLK MODE | INSTR | MUL PIPE | ALU PIPE | P REG | C REG | S REG | Y BUS | COMMENT |
|-----|---------------------|---------------------|-----------|-----------------|-------------|-------------|-------------|-------------|----------|----------|----------|----------|---------------------------|
| 19 | c ₁₄ MSH | c ₁₄ LSH | 2.0 | c ₁₄ | 1 | SR19•RB19 | | | | | S6 | | Start core calculation |
| 20 | | | 2.0 | c ₁₄ | 0 | PR21 + RB21 | SR19•RB19 | | | S6 | | | S7 is input to core calc. |
| 21 | c ₁₃ MSH | c ₁₃ LSH | 2.0 | c ₁₃ | 0 | PR21 + RB21 | | | P6 | | | | |
| 22 | | | 2.0 | c ₁₃ | 1 | SR22•CR22 | | | | | S7 | | |
| 23 | | | 2.0 | c ₁₃ | 0 | PR24 + RB24 | SR22•CR22 | | | | | | |
| 24 | c ₁₂ MSH | c ₁₂ LSH | 2.0 | c ₁₂ | 0 | PR24 + RB24 | | | P7 | | | | |
| 25 | | | 2.0 | c ₁₂ | 1 | SR25•CR25 | | | | | S8 | | |
| 26 | | | 2.0 | c ₁₂ | 0 | PR27 + RB27 | SR25•CR25 | | | | | | |
| 27 | c ₁₁ MSH | c ₁₁ LSH | 2.0 | c ₁₁ | 0 | PR27 + RB27 | | | P8 | | | | |
| 28 | | | 2.0 | c ₁₁ | 1 | SR28•CR28 | | | | | S9 | | |
| 29 | | | 2.0 | c ₁₁ | 0 | PR30 + RB30 | SR28•CR28 | | | | | | |
| 30 | c ₁₀ MSH | c ₁₀ LSH | 2.0 | c ₁₀ | 0 | PR30 + RB30 | | | P9 | | | | |
| 31 | | | 2.0 | c ₁₀ | 1 | SR31•CR31 | | | | | S10 | | |
| 32 | | | 2.0 | c ₁₀ | 0 | PR33 + RB33 | SR31•CR31 | | | | | | |
| 33 | c ₉ MSH | c ₉ LSH | 2.0 | c ₉ | 0 | PR33 + RB33 | | | P10 | | | | |
| 34 | | | 2.0 | c ₉ | 1 | SR34•CR34 | | | | | S11 | | |
| 35 | | | 2.0 | c ₉ | 0 | PR36 + RB36 | SR34•CR34 | | | | | | |
| 36 | c ₈ MSH | c ₈ LSH | 2.0 | c ₈ | 0 | PR36 + RB36 | | | P11 | | | | |
| 37 | | | 2.0 | c ₈ | 1 | SR37•CR37 | | | | | S12 | | |
| 38 | | | 2.0 | c ₈ | 0 | PR39 + RB39 | SR37•CR37 | | | | | | |
| 39 | c ₇ MSH | c ₇ LSH | 2.0 | c ₇ | 0 | PR39 + RB39 | | | P12 | | | | |
| 40 | | | 2.0 | c ₇ | 1 | SR40•CR40 | | | | | S13 | | |
| 41 | | | 2.0 | c ₇ | 0 | PR42 + RB42 | SR40•CR40 | | | | | | |
| 42 | c ₆ MSH | c ₆ LSH | 2.0 | c ₆ | 0 | PR42 + RB42 | | | P13 | | | | |

Table 9. Pseudocode for Chebyshev Tangent Routine (PIPES2-0 = 010, RND1-0 = 00) (Continued)

| CLK | DA BUS | DB BUS | RA REG | RB REG | CLK MODE | INSTR | MUL PIPE | ALU PIPE | P REG | C REG | S REG | Y BUS | COMMENT |
|-----|--------------------|--------------------|-----------|----------------|-------------|-----------|-------------|-------------|----------|----------|----------|----------|---|
| 43 | | | 2.0 | c ₆ | 1 | SR43*CR43 | | | | | S14 | | |
| 44 | | | 2.0 | c ₆ | 0 | PR45+RB45 | SR43*CR43 | | | | | | |
| 45 | c ₅ MSH | c ₅ LSH | 2.0 | c ₅ | 0 | PR45+RB45 | | | P14 | | | | |
| 46 | | | 2.0 | c ₅ | 1 | SR46*CR46 | | | | | S15 | | |
| 47 | | | 2.0 | c ₅ | 0 | PR48+RB48 | SR46*CR46 | | | | | | |
| 48 | c ₄ MSH | c ₄ LSH | 2.0 | c ₄ | 0 | PR48+RB48 | | | P15 | | | | |
| 49 | | | 2.0 | c ₄ | 1 | SR49*CR49 | | | | | S16 | | |
| 50 | | | 2.0 | c ₄ | 0 | PR51+RB51 | SR49*CR49 | | | | | | |
| 51 | c ₃ MSH | c ₃ LSH | 2.0 | c ₃ | 0 | PR51+RB51 | | | P16 | | | | |
| 52 | | | 2.0 | c ₃ | 1 | SR52*CR52 | | | | | S17 | | |
| 53 | | | 2.0 | c ₃ | 0 | PR54+RB54 | SR52*CR52 | | | | | | |
| 54 | c ₂ MSH | c ₂ LSH | 2.0 | c ₂ | 0 | PR54+RB54 | | | P17 | | | | |
| 55 | | | 2.0 | c ₂ | 1 | SR55*CR55 | | | | | S18 | | |
| 56 | | | 2.0 | c ₂ | 0 | PR57+RB57 | SR55*CR55 | | | | | | |
| 57 | c ₁ MSH | c ₁ LSH | 2.0 | c ₁ | 0 | PR57+RB57 | | | P18 | | | | |
| 58 | | | 2.0 | c ₁ | 1 | SR58*CR58 | | | | | S19 | | |
| 59 | | | 2.0 | c ₁ | 0 | PR60+RB60 | SR58*CR58 | | | | | | |
| 60 | c ₀ MSH | c ₀ LSH | 2.0 | c ₀ | 0 | PR60+RB60 | | | P19 | | | | |
| 61 | S5 MSH | S5 LSH | 2.0 | S5 | 1 | SR61*RB61 | | | | | S20 | | Begin postprocessing |
| 62 | | | 2.0 | S5 | 0 | DUMMY | SR61*RB61 | | | | | | Instruction is RA+RB, used to allow time for result to propagate to Y bus |

Table 9. Pseudocode for Chebyshev Tangent Routine (PIPES2-0 = 010, RND1-0 = 00) (Continued)

Table 9. Pseudocode for Chebyshev Tangent Routine (PIPES2-0 = 010, RND1-0 = 00) (Concluded)

| CLK | DA BUS | DB BUS | RA REG | RB REG | CLK MODE | INSTR | MUL PIPE | ALU PIPE | P REG | C REG | S REG | Y BUS | COMMENT |
|-----|-----------|-----------|-----------|-----------|-------------|-------|-------------|-------------|----------|----------|----------|----------|--|
| 63 | | | 2.0 | S5 | 0 | NOP | | | P20 | | | P20 | Output MSH, if cycle 13b was executed then P20 is the answer; if cycle 13a was executed then the answer is 1.0/P20, which is calculated next |
| 64 | 1.0 MSH | 1.0 LSH | 2.0 | S5 | 0 | DIV | | | | | | P20 | Output LSH |
| 65 | P20 MSH | P20 LSH | 1.0 | P20 | 0 | DIV | | | | | | | Operands for Division must come from RA and RB, feedback is not an option |
| 66 | | | 1.0 | P20 | 0 | NOP | | | | | | | Wait for Division result |
| 67 | | | 1.0 | P20 | 0 | NOP | | | | | | | Wait for Division result |
| 68 | | | 1.0 | P20 | 0 | NOP | | | | | | | Wait for Division result |
| 69 | | | 1.0 | P20 | 0 | NOP | | | | | | | Wait for Division result |
| 70 | | | 1.0 | P20 | 0 | NOP | | | | | | | Wait for Division result |
| 71 | | | 1.0 | P20 | 0 | NOP | | | | | | | Wait for Division result |
| 72 | | | 1.0 | P20 | 0 | NOP | | | | | | | Wait for Division result |
| 73 | | | 1.0 | P20 | 0 | NOP | | | | | | | Wait for Division result |
| 74 | | | 1.0 | P20 | 0 | NOP | | | | | | | Wait for Division result |
| 75 | | | 1.0 | P20 | 0 | NOP | | | | | | | Wait for Division result |
| 76 | | | 1.0 | P20 | 0 | NOP | | | | | | | Wait for Division result |
| 77 | | | 1.0 | P20 | 0 | NOP | | | | | | | Wait for Division result |
| 78 | | | 1.0 | P20 | 0 | NOP | | | P21 | | | P21 | Output MSH of answer |
| 79 | | | 1.0 | P20 | 0 | NOP | | | P21 | | | P21 | Output LSH of answer |

Table 9. Pseudocode for Chebyshev Tangent Routine (PIPES2-0 = 010, RND1-0 = 00) (Continued)

Microcode Table for the Tangent(x) Calculation

All numbers are in hex. Any field with a length that is not a multiple of 4 is right justified and zero filled. For the microcode table, the value of X has been chosen to be 1/3 pi.

| P A | D A | D B | P B | E N | E N | C L | P I | C L | C O | S E | R E | H A | E N | F L | I N | R N | F S | S B | S T | S T | S E | O E | O E |
|------------|----------|---------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| | | | A B | A B | K C | P E | M F | O I | P D | G | S L | C O | S R | D S | C T | L S | L Y | S C | | | | | |
| F 3FF0C152 | 382D7365 | F 0 0 _ | 2 0 3 | FF 1 | 1 1 | 1 0 | 1C0 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | | | | | |
| F 3FF45F30 | 6DC9C883 | F 1 1 _ | 2 0 3 | FF 1 | 1 1 | 1 0 | 1C0 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | | | | | | |
| F 00000000 | 00000000 | F 0 0 _ | 2 0 3 | FB 1 | 1 1 | 1 0 | 180 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | | | | | | |
| F 3FF00000 | 00000000 | F 0 1 _ | 2 0 3 | FB 1 | 1 1 | 1 0 | 180 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | | | | | | |
| F 3FD00000 | 00000000 | F 0 1 J | 2 1 3 | BF 1 | 1 1 | 0 0 | 1C0 | 0 0 | 1 0 | 3 3 | 1 0 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | | | | | | |
| F 3FF00000 | 00000000 | F 0 0 _ | 2 0 3 | FB 1 | 1 1 | 1 0 | 1A3 | 1 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | | | | | | |
| F 00000000 | 00000000 | F 1 0 _ | 2 0 3 | FB 1 | 1 1 | 1 0 | 1A3 | 1 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | | | | | | |
| F 00000000 | 00000004 | F 0 1 _ | 2 0 1 | BF 1 | 1 1 | 1 0 | 240 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | | | | | | |
| F 00000000 | 00000000 | F 0 0 _ | 2 1 3 | FB 1 | 1 1 | 1 0 | 1A2 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | | | | | | |
| F 00000000 | 00000000 | F 0 0 _ | 2 1 3 | F6 1 | 1 1 | 1 0 | 181 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | | | | | | |
| F 00000000 | 00000000 | F 0 0 _ | 2 1 3 | FE 1 | 1 1 | 1 0 | 182 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | | | | | | |
| F 00000000 | 00000000 | F 0 0 J | 2 0 3 | FF 1 | 1 1 | 0 0 | 300 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | | | | | | |
| F 40000000 | 00000000 | F 0 1 _ | 2 1 3 | F7 1 | 1 1 | 1 0 | 183 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | | | | | | |
| F 00000000 | 00000000 | F 0 0 _ | 2 1 3 | 5F 1 | 1 1 | 1 0 | 1C0 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | | | | | | |
| F 40000000 | 00000000 | F 0 0 J | 2 0 3 | EF 1 | 1 1 | 0 0 | 1C0 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | | | | | | |
| F 00000000 | 00000000 | F 1 0 _ | 2 0 3 | EF 1 | 1 1 | 1 0 | 1C0 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | | | | | | |
| F 00000000 | 00000000 | F 0 0 _ | 2 0 3 | FB 1 | 1 1 | 1 0 | 180 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | | | | | | |
| F BFF00000 | 00000000 | F 0 1 _ | 2 0 3 | FB 1 | 1 1 | 1 0 | 180 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | | | | | | |
| F 3D747D84 | 2210CC35 | F 0 1 _ | 2 1 3 | BF 1 | 1 1 | 1 0 | 1C0 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | | | | | | |
| F 00000000 | 00000000 | F 0 0 J | 2 0 3 | FB 1 | 1 1 | 0 0 | 180 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | | | | | | |
| F 3DA1D666 | 36043991 | F 0 1 _ | 2 0 3 | FB 1 | 1 1 | 1 0 | 180 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | 3 3 | 1 0 | 0 0 | 0 0 | | | | | | |

Microcode Table for the Tangent(x) Calculation (Continued)

| P | D | D | P | E | E | C | P | C | C | S | R | H | E | F | I | R | F | S | B | S | T | S | O | O | O |
|---|----------|----------|---|---|---|---|---|---|---|----|---|---|---|---|-----|---|---|---|---|---|---|---|---|---|---|
| A | A | B | B | N | N | L | I | L | O | E | E | A | N | L | N | N | A | R | Y | E | E | E | E | E | E |
| | | | A | B | K | P | K | N | L | S | L | C | O | S | D | S | C | T | L | S | L | Y | S | C | |
| | | | | | C | E | M | F | O | E | T | | W | T | | | | T | C | E | S | T | Y | | |
| | | | | | | S | O | I | P | T | | | C | R | | | | | | P | T | | | | |
| | | | | | | D | G | | | | | | | | | | | | | | | | | | |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 3DCCD078 | F52B3A73 | F | 0 | 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 3DF938F9 | CDDFF864 | F | 0 | 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 3E262043 | 0E99B5B7 | F | 0 | 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 3E535C2C | 953CE515 | F | 0 | 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 3E80F07A | FC099D7F | F | 0 | 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 3EADA4D7 | 89EB45C4 | F | 0 | 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 3ED9F03D | 4C51A771 | F | 0 | 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 3F06B236 | DE4D014C | F | 0 | 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |

Microcode Table for the Tangent(x) Calculation (Continued)

| P A | D A | D B | P B | E N A | E N B | C L K | P P | C L K | C N F | S E O | R E S | H A N | E L C | F O W | I N S | R N D | F S C | S A R | B Y E | S T L | S E E | O E E | O E E | O E E | |
|--------|--------|--------|--------|-------------|-------------|-------------|--------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|--|
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | |

Microcode Table for the Tangent(x) Calculation (Concluded)

| P A | D A | D B | P B | E N A | E N B | C L K | P I P | C L O K N | S E L | R E S | H E L C | F L O | I N S T R | R N D | F N A R D | S C T | B Y T E S | S T E E L Y | S E E L Y | O O O | O O O | | | | |
|--------|----------|----------|--------|-------------|-------------|-------------|-------------|-----------------------|-------------|-------------|------------------|-------------|-----------------------|-------------|-----------------------|-------------|-----------------------|----------------------------|-----------------------|-------------|-------------|---|---|---|---|
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 0 | 3 | FF | 1 | 1 | 1 | 0 | 300 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 0 | 3 | FF | 1 | 1 | 1 | 0 | 300 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 0 | 3 | FF | 1 | 1 | 1 | 0 | 300 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 0 | 3 | FF | 1 | 1 | 1 | 0 | 300 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 0 | 3 | FF | 1 | 1 | 1 | 0 | 300 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 0 | 3 | FF | 1 | 1 | 1 | 0 | 300 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 0 | 3 | FF | 1 | 1 | 1 | 0 | 300 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 0 | 3 | FF | 1 | 1 | 1 | 0 | 300 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 0 | 3 | FF | 1 | 1 | 1 | 0 | 300 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 0 | 3 | FF | 1 | 1 | 1 | 0 | 300 | 0 | 0 | 0 | 0 | 3 | 3 | 0 | 0 | 0 | 0 |
| F | 3E9180DE | 301E3V23 | F | 0 | 1 | — | 3 | 0 | 3 | EB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 3 | 0 | 3 | EB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 3 | 1 | 3 | 8E | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 3E33DBE8 | 01B3E412 | F | 0 | 1 | — | 3 | 0 | 3 | EB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 3 | 0 | 3 | EB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 3 | 1 | 3 | 8E | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |

ArcSine and ArcCosine Routine Using Chebyshev's Method

All floating-point inputs and outputs are double precision. The output is in radians.

Steps Required to Perform the Calculation

STEP 1 — Preprocessing; range reduction is not needed, because an input, X , outside the range of $[-1, 1]$ indicates an error. This routine requires that the X^2 be less than or equal to $1/2$. The first operation to be performed is to square X , then multiply it by 4.0 , and finally subtract 1.0 .

$$X1 \leftarrow X * X * 4 - 1$$

STEP 2 — Core Calculation; $X1$ in Step 1 will be referred to as ' x ' in the core calculation.

$$X2 \leftarrow C_{\text{series_asin\&acos}}$$

$$\begin{aligned} \leftarrow & (((((((((((((c18 * x + c17) * x + c16) * x + \\ & c15 * x + c14) * x + c13) * x + c12) * x + c11) * x + c10) * x + \\ & c9) * x + c8) * x + c7) * x + c6) * x + c5) * x + c4) * x + c3) * x + \\ & c2) * x + c1) * x + c0 \end{aligned}$$

STEP 3 — Postprocessing; multiply the output of the core calculation times $\text{SQRT}(2.0)$, then multiply this product by X , the original input. This yields $\text{ArcSine}(X)$. To calculate $\text{ArcCosine}(X)$, the following identity is used:

$$\text{ArcCosine}(X) = \pi/2 - \text{ArcSine}(X)$$

$$X3 \leftarrow X2 * \text{SQRT}(2.0)$$

$$\text{ArcSine}(X) \leftarrow X3 * X$$

$$\text{ArcCosine}(X) \leftarrow \pi/2 - \text{ArcSine}(X)$$

Algorithms for the Three Steps

Step 1 perform the preprocessing:

$$T1 \leftarrow X * X$$

$$T2 \leftarrow 4.0 * T1$$

$$T3 \leftarrow T2 - 1$$

$T3$ is $X1$ in Step 1, the input to the core routine

Step 2 perform the core calculation:

```

T4 ← c18*CREG
T5 ← T4 + c17
T6 ← T5*CREG
T7 ← T6 + c16
T8 ← T7*CREG
T9 ← T8 + c15
T10 ← T9*CREG
T11 ← T10 + c14
T12 ← T11*CREG
T13 ← T12 + c13
T14 ← T13*CREG
T15 ← T14 + c12
T16 ← T15*CREG
T17 ← T16 + c11
T18 ← T17*CREG
T19 ← T18 + c10
T20 ← T19*CREG
T21 ← T20 + c9
T22 ← T21*CREG
T23 ← T22 + c8
T24 ← T23*CREG
T25 ← T24 + c7
T26 ← T25*CREG
T27 ← T26 + c6
T28 ← T27*CREG
T29 ← T28 + c5
T30 ← T29*CREG
T31 ← T30 + c4
T32 ← T31*CREG
T33 ← T32 + c3
T34 ← T33*CREG
T35 ← T34 + c2
T36 ← T35*CREG
T37 ← T36 + c1
T38 ← T37*CREG
T39 ← T38 + c0

```

CREG ← T3

Step 3 perform the postprocessing:

```

T40 ← X*T39
ArcSine(X) ← T40*SQRT(2.0)
ArcCosine(X) ← pi/2 - ArcSine(X)

```

SQRT(2.0) entered as a constant

Required System Intervention

There is no system intervention required to calculate ArcSine(X) and ArcCosine(X).

Number of 'ACT8847' Cycles Required to Calculate ArcSine(x) and ArcCosine(x)

The total number of cycles required to perform the ArcSine(x) and ArcCosine(x) calculation is 68.

Listing of the Chebyshev Constants (c's)

The constants are represented in IEEE double-precision floating-point format.

c18 = 3DA4A49F8CCD9E73
c17 = 3DC05DFE52AAD200
c16 = 3DCCF31E26F94C8D
c15 = 3DE86CDA3C8CAEB0
c14 = 3E0768D9F4E950EA
c13 = 3E2383A37598FC80
c12 = 3E403E4B2F65F0DE
c11 = 3E5BAFC8245ABDF8
c10 = 3E77E3333AFF1AB4
c9 = 3E94E3A4D4220C9C
c8 = 3EB296DD4C084ACB
c7 = 3ED0E913F5F9D496
c6 = 3EEFA74E896F8FA8
c5 = 3F0EC76B7832DBB6
c4 = 3F2F978698C8B2E4
c3 = 3F519B1087542073
c2 = 3F7696895FFC05A0
c1 = 3FA375CA61D2988C
c0 = 3FE7B20423D1D930

Pseudocode Table for the ArcSine(x) and ArcCosine(x) Calculation

Table 10. Pseudocode for Chebyshev ArcSine and ArcCosine Routine (PIPES2-0 = 010, RND1-0 = 00)

| CLK | DA BUS | DB BUS | RA REG | RB REG | CLK MODE | INSTR | MUL PIPE | ALU PIPE | P REG | C REG | S REG | Y BUS | COMMENT |
|-----|---------------------|---------------------|-----------|-----------------|-------------|-----------|-------------|-------------|----------|----------|----------|----------|---------------------------|
| 1 | X MSH | X LSH | | | 0 | RA2*RB2 | | | | | | | X is the input |
| 2 | X MSH | X LSH | X | X | 0 | RA2*RB2 | | | | | | | |
| 3 | 4.0 MSH | 4.0 LSH | X | X | 0 | RA4*PR4 | RA2*RB2 | | | | | | |
| 4 | | | 4.0 | X | 0 | RA4*PR4 | | | P1 | | | | |
| 5 | | | 4.0 | X | 0 | PR6+RB6 | RA4*PR4 | | | | | | |
| 6 | -1.0 MSH | -1.0 LSH | 4.0 | -1.0 | 0 | PR6+RB6 | | | P2 | | | | |
| 7 | c ₁₈ MSH | c ₁₈ LSH | 4.0 | c ₁₈ | 1 | SR7*RB7 | | | | | S1 | | Start core calculation |
| 8 | | | 4.0 | c ₁₈ | 0 | PR9+RB9 | SR7*RB7 | | | S1 | | | S1 is input to core calc. |
| 9 | c ₁₇ MSH | c ₁₇ LSH | 4.0 | c ₁₇ | 0 | PR9+RB9 | | | P3 | | | | |
| 10 | | | 4.0 | c ₁₇ | 1 | SR10*CR10 | | | | | S2 | | |
| 11 | | | 4.0 | c ₁₇ | 0 | PR12+RB12 | SR10*CR10 | | | | | | |
| 12 | c ₁₆ MSH | c ₁₆ LSH | 4.0 | c ₁₆ | 0 | PR12+RB12 | | | P4 | | | | |
| 13 | | | 4.0 | c ₁₆ | 1 | SR13*CR13 | | | | | S3 | | |
| 14 | | | 4.0 | c ₁₆ | 0 | PR15+RB15 | SR13*CR13 | | | | | | |
| 15 | c ₁₅ MSH | c ₁₅ LSH | 4.0 | c ₁₅ | 0 | PR15+RB15 | | | P5 | | | | |
| 16 | | | 4.0 | c ₁₅ | 1 | SR16*CR16 | | | | | S4 | | |
| 17 | | | 4.0 | c ₁₅ | 0 | PR18+RB18 | SR16*CR16 | | | | | | |
| 18 | c ₁₄ MSH | c ₁₄ LSH | 4.0 | c ₁₄ | 0 | PR18+RB18 | | | P6 | | | | |
| 19 | | | 4.0 | c ₁₄ | 1 | SR19*CR19 | | | | | S5 | | |
| 20 | | | 4.0 | c ₁₄ | 0 | PR21+RB21 | SR19*CR19 | | | | | | |
| 21 | c ₁₃ MSH | c ₁₃ LSH | 4.0 | c ₁₃ | 0 | PR21+RB21 | | | P7 | | | | |
| 22 | | | 4.0 | c ₁₃ | 1 | SR22*CR22 | | | | | S6 | | |
| 23 | | | 4.0 | c ₁₃ | 0 | PR24+RB24 | SR22*CR22 | | | | | | |

Table 10. Pseudocode for Chebyshev ArcSine and ArcCosine Routine (PIPES2-0 = 010, RND1-0 = 00) (Continued)

| CLK | DA BUS | DB BUS | RA REG | RB REG | CLK MODE | INSTR | MUL PIPE | ALU PIPE | P REG | C REG | S REG | Y BUS | COMMENT |
|-----|---------------------|---------------------|-----------|-----------------|-------------|-------------|-------------|-------------|----------|----------|----------|----------|---------|
| 24 | c ₁₂ MSH | c ₁₂ LSH | 4.0 | c ₁₂ | 0 | PR24 + RB24 | | | P8 | | | | |
| 25 | | | 4.0 | c ₁₂ | 1 | SR25 * CR25 | | | | | S7 | | |
| 26 | | | 4.0 | c ₁₂ | 0 | PR27 + RB27 | SR25 * CR25 | | | | | | |
| 27 | c ₁₁ MSH | c ₁₁ LSH | 4.0 | c ₁₁ | 0 | PR27 + RB27 | | | P9 | | | | |
| 28 | | | 4.0 | c ₁₁ | 1 | SR28 * CR28 | | | | | S8 | | |
| 29 | | | 4.0 | c ₁₁ | 0 | PR30 + RB30 | SR28 * CR28 | | | | | | |
| 30 | c ₁₀ MSH | c ₁₀ LSH | 4.0 | c ₁₀ | 0 | PR30 + RB30 | | | P10 | | | | |
| 31 | | | 4.0 | c ₁₀ | 1 | SR31 * CR31 | | | | | S9 | | |
| 32 | | | 4.0 | c ₁₀ | 0 | PR33 + RB33 | SR31 * CR31 | | | | | | |
| 33 | c ₉ MSH | c ₉ LSH | 4.0 | c ₉ | 0 | PR33 + RB33 | | | P11 | | | | |
| 34 | | | 4.0 | c ₉ | 1 | SR34 * CR34 | | | | | S10 | | |
| 35 | | | 4.0 | c ₉ | 0 | PR36 + RB36 | SR34 * CR34 | | | | | | |
| 36 | c ₈ MSH | c ₈ LSH | 4.0 | c ₈ | 0 | PR36 + RB36 | | | P12 | | | | |
| 37 | | | 4.0 | c ₈ | 1 | SR37 * CR37 | | | | | S11 | | |
| 38 | | | 4.0 | c ₈ | 0 | PR39 + RB39 | SR37 * CR37 | | | | | | |
| 39 | c ₇ MSH | c ₇ LSH | 4.0 | c ₇ | 0 | PR39 + RB39 | | | P13 | | | | |
| 40 | | | 4.0 | c ₇ | 1 | SR40 * CR40 | | | | | S12 | | |
| 41 | | | 4.0 | c ₇ | 0 | PR42 + RB42 | SR40 * CR40 | | | | | | |
| 42 | c ₆ MSH | c ₆ LSH | 4.0 | c ₆ | 0 | PR42 + RB42 | | | P14 | | | | |
| 43 | | | 4.0 | c ₆ | 1 | SR43 * CR43 | | | | | S13 | | |
| 44 | | | 4.0 | c ₆ | 0 | PR45 + RB45 | SR43 * CR43 | | | | | | |
| 45 | c ₅ MSH | c ₅ LSH | 4.0 | c ₅ | 0 | PR45 + RB45 | | | P15 | | | | |
| 46 | | | 4.0 | c ₅ | 1 | SR46 * CR46 | | | | | S14 | | |
| 47 | | | 4.0 | c ₅ | 0 | PR48 + RB48 | SR46 * CR46 | | | | | | |
| 48 | c ₄ MSH | c ₄ LSH | 4.0 | c ₄ | 0 | PR48 + RB48 | | | P16 | | | | |

Table 10. Pseudocode for Chebyshev ArcSine and ArcCosine Routine (PIPES2-0 = 010, RND1-0 = 00) (Concluded)

| CLK | DA BUS | DB BUS | RA REG | RB REG | CLK MODE | INSTR | MUL PIPE | ALU PIPE | P REG | C REG | S REG | Y BUS | COMMENT |
|-----|--------------------|--------------------|-----------|----------------|-------------|-----------|-------------|-------------|----------|----------|----------|----------|---|
| 49 | | | 4.0 | c ₄ | 1 | SR49•CR49 | | | | | S15 | | |
| 50 | | | 4.0 | c ₄ | 0 | PR51+RB51 | SR49•CR49 | | | | | | |
| 51 | c ₃ MSH | c ₃ LSH | 4.0 | c ₃ | 0 | PR51+RB51 | | | P17 | | | | |
| 52 | | | 4.0 | c ₃ | 1 | SR52•CR52 | | | | | S16 | | |
| 53 | | | 4.0 | c ₃ | 0 | PR54+RB54 | SR52•CR52 | | | | | | |
| 54 | c ₂ MSH | c ₂ LSH | 4.0 | c ₂ | 0 | PR54+RB54 | | | P18 | | | | |
| 55 | | | 4.0 | c ₂ | 1 | SR55•CR55 | | | | | S17 | | |
| 56 | | | 4.0 | c ₂ | 0 | PR57+RB57 | SR55•CR55 | | | | | | |
| 57 | c ₁ MSH | c ₁ LSH | 4.0 | c ₁ | 0 | PR57+RB57 | | | P19 | | | | |
| 58 | | | 4.0 | c ₁ | 1 | SR58•CR58 | | | | | S18 | | |
| 59 | | | 4.0 | c ₁ | 0 | PR60+RB60 | SR58•CR58 | | | | | | |
| 60 | c ₀ MSH | c ₀ LSH | 4.0 | c ₀ | 0 | PR60+RB60 | | | P20 | | | | |
| 61 | X MSH | X LSH | 4.0 | X | 1 | SR61•RB61 | | | | | S19 | | Begin postprocessing |
| 62 | SQRT(2) MSH | SQRT(2) LSH | 4.0 | X | 0 | RA63•PR63 | SR61•RB61 | | | | | | SQRT(2) is the real value of square root of 2.0 |
| 63 | | | SQRT 2 | X | 0 | RA63•PR63 | | | P21 | | | | |
| 64 | | | SQRT 2 | X | 0 | DUMMY | RA63•PR63 | | | | | | Instruction is double- precision RA + RB, prevents ArcCosine from over- writing ArcSine result |
| 66 | pi/2 MSH | pi/2 LSH | SQRT 2 | pi/2 | 1 | RB66-PR66 | | | P22 | | | P22 | Output LSH of ArcSine |
| 67 | | | SQRT 2 | pi/2 | 0 | NOP | | | | | S20 | S20 | Output MSH of ArcCosine |
| 68 | | | SQRT 2 | pi/2 | 0 | NOP | | | | | S20 | S20 | Output LSH of ArcCosine |

Microcode Table for the ArcSine(x) and ArcCosine(x) Calculation

All numbers are in hex. Any field with a length that is not a multiple of 4 is right justified and zero filled. For the microcode table, the value of X has been chosen to be 1/(SQRT(2.0)).

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P | D | D | P | E | E | C | P | C | C | S | R | H | E | F | I | R | F | S | B | S | T | S | O | O | O |
| A | A | B | B | N | N | L | I | L | O | E | E | A | N | L | N | N | A | R | Y | E | E | E | E | E | E |
| | | | A | B | K | | P | K | N | L | S | L | C | O | S | D | S | C | T | L | S | L | Y | S | C |
| | | | | | C | | E | M | F | O | E | T | | W | T | | T | C | E | S | T | Y | | | |
| | | | | | | | S | O | I | P | T | | | C | R | | | | P | T | | | | | |
| | | | | | | | | D | G | | | | | | | | | | | | | | | | |

| | | | | | | | | | |
|------------|----------|---------|-------|----|-------|-------|---------|-------|-------|
| F 3FE6A09E | 667F3BCD | F 0 0 _ | 2 0 3 | FF | 1 1 1 | 0 1C0 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 3FE6A09E | 667F3BCD | F 1 1 _ | 2 0 3 | FF | 1 1 1 | 0 1C0 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 40100000 | 00000000 | F 0 0 _ | 2 0 3 | EF | 1 1 1 | 0 1C0 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 00000000 | 00000000 | F 1 0 _ | 2 0 3 | EF | 1 1 1 | 0 1C0 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 00000000 | 00000000 | F 0 0 _ | 2 0 3 | FB | 1 1 1 | 0 180 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F BFF00000 | 00000000 | F 0 1 _ | 2 0 3 | FB | 1 1 1 | 0 180 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 3DA4A49F | 8CCD9E73 | F 0 1 _ | 2 1 3 | BF | 1 1 1 | 0 1C0 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 00000000 | 00000000 | F 0 0 _ | 2 0 3 | FB | 1 1 0 | 0 180 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 3DC05DFE | 52AAD200 | F 0 1 _ | 2 0 3 | FB | 1 1 1 | 0 180 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 00000000 | 00000000 | F 0 0 _ | 2 1 3 | 9F | 1 1 1 | 0 1C0 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 00000000 | 00000000 | F 0 0 _ | 2 0 3 | FB | 1 1 1 | 0 180 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 3DCCF31E | 26F94C8D | F 0 1 _ | 2 0 3 | FB | 1 1 1 | 0 180 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 00000000 | 00000000 | F 0 0 _ | 2 1 3 | 9F | 1 1 1 | 0 1C0 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 00000000 | 00000000 | F 0 0 _ | 2 0 3 | FB | 1 1 1 | 0 180 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 3DE86CDA | 3C8CAEB0 | F 0 1 _ | 2 0 3 | FB | 1 1 1 | 0 180 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 00000000 | 00000000 | F 0 0 _ | 2 1 3 | 9F | 1 1 1 | 0 1C0 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 00000000 | 00000000 | F 0 0 _ | 2 0 3 | FB | 1 1 1 | 0 180 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 3E0768D9 | F4E950EA | F 0 1 _ | 2 0 3 | FB | 1 1 1 | 0 180 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 00000000 | 00000000 | F 0 0 _ | 2 1 3 | 9F | 1 1 1 | 0 1C0 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 00000000 | 00000000 | F 0 0 _ | 2 0 3 | FB | 1 1 1 | 0 180 | 0 0 0 0 | 3 3 1 | 0 0 0 |

Microcode Table for the ArcSine(x) and ArcCosine(x) Calculation (Continued)

Microcode Table for the ArcSine(x) and ArcCosine(x) Calculation (Continued)

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P | D | D | P | E | E | C | P | C | C | S | R | H | E | F | I | R | F | S | B | S | T | S | O | O | O |
| A | A | B | B | N | N | L | I | L | O | E | E | A | N | L | N | N | A | R | Y | E | E | E | E | E | E |
| | | | A | B | K | P | K | N | L | S | L | C | O | S | D | S | C | T | L | S | L | Y | S | C | |
| | | | | | C | E | M | F | O | E | T | | W | T | | T | C | E | S | T | Y | | | | |
| | | | | | S | O | I | P | T | | | | C | R | | | | P | T | | | | | | |
| | | | | | | D | G | | | | | | | | | | | | | | | | | | |

| | | | | | | | | |
|------------|----------|---------|-------|----------|-------|---------|-------|-------|
| F 3E2383A3 | 7598FC80 | F 0 1 — | 2 0 3 | FB 1 1 1 | 0 180 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 00000000 | 00000000 | F 0 0 — | 2 1 3 | 9F 1 1 1 | 0 1C0 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 00000000 | 00000000 | F 0 0 — | 2 0 3 | FB 1 1 1 | 0 180 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 3E403E4B | 2F65F0DE | F 0 1 — | 2 0 3 | FB 1 1 1 | 0 180 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 00000000 | 00000000 | F 0 0 — | 2 1 3 | 9F 1 1 1 | 0 1C0 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 00000000 | 00000000 | F 0 0 — | 2 0 3 | FB 1 1 1 | 0 180 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 3E5BAFC8 | 245ABDF8 | F 0 1 — | 2 0 3 | FB 1 1 1 | 0 180 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 00000000 | 00000000 | F 0 0 — | 2 1 3 | 9F 1 1 1 | 0 1C0 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 00000000 | 00000000 | F 0 0 — | 2 0 3 | FB 1 1 1 | 0 180 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 3E77E333 | 3AFF1AB4 | F 0 1 — | 2 0 3 | FB 1 1 1 | 0 180 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 00000000 | 00000000 | F 0 0 — | 2 1 3 | 9F 1 1 1 | 0 1C0 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 00000000 | 00000000 | F 0 0 — | 2 0 3 | FB 1 1 1 | 0 180 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 3E94E3A4 | D4220C9C | F 0 1 — | 2 0 3 | FB 1 1 1 | 0 180 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 00000000 | 00000000 | F 0 0 — | 2 1 3 | 9F 1 1 1 | 0 1C0 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 00000000 | 00000000 | F 0 0 — | 2 0 3 | FB 1 1 1 | 0 180 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 3EB296DD | 4C084ACB | F 0 1 — | 2 0 3 | FB 1 1 1 | 0 180 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 00000000 | 00000000 | F 0 0 — | 2 1 3 | 9F 1 1 1 | 0 1C0 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 00000000 | 00000000 | F 0 0 — | 2 0 3 | FB 1 1 1 | 0 180 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 3ED0E913 | F5F9D496 | F 0 1 — | 2 0 3 | FB 1 1 1 | 0 180 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 00000000 | 00000000 | F 0 0 — | 2 1 3 | 9F 1 1 1 | 0 1C0 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 00000000 | 00000000 | F 0 0 — | 2 0 3 | FB 1 1 1 | 0 180 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 3EEFA74E | 896F8FA8 | F 0 1 — | 2 0 3 | FB 1 1 1 | 0 180 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 00000000 | 00000000 | F 0 0 — | 2 1 3 | 9F 1 1 1 | 0 1C0 | 0 0 0 0 | 3 3 1 | 0 0 0 |
| F 00000000 | 00000000 | F 0 0 — | 2 0 3 | FB 1 1 1 | 0 180 | 0 0 0 0 | 3 3 1 | 0 0 0 |

Microcode Table for the ArcSine(x) and ArcCosine(x) Calculation (Concluded)

| P A | D A | D B | P B | E N A | E N B | C L K C | P I P S | C C L O M F O I D G | S E L S E T | R̄ H̄ E F I R N F S B S T S Ō Ē Ō | Ĥ Ĥ A N L C O W C R | R N D | F S D T | S B R Y C C E S T P T | S T E E L S Y | S̄ Ō Ē Ō | S̄ Ē Ō Ē Ō | S̄ Ē Ō Ē Ō |
|------------|----------|--------|--------|-------------|-------------|------------------|------------------|--|----------------------------|---|--|-------------|------------------|---|---------------------------------|----------------------|----------------------------|----------------------------|
| F 3F0EC76B | 7832DBB6 | F 0 1 | — | 2 | 0 | 3 | FB | 1 1 1 | 0 | 180 | 0 0 0 | 0 3 3 | 1 0 0 | 0 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 1 1 | 0 | 1C0 | 0 0 0 | 0 3 3 | 1 0 0 | 0 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 1 1 | 0 | 180 | 0 0 0 | 0 3 3 | 1 0 0 | 0 | 0 | 0 | 0 | 0 |
| F 3F2F9786 | 98C8B2E4 | F 0 1 | — | 2 | 0 | 3 | FB | 1 1 1 | 0 | 180 | 0 0 0 | 0 3 3 | 1 0 0 | 0 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 1 1 | 0 | 1C0 | 0 0 0 | 0 3 3 | 1 0 0 | 0 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 1 1 | 0 | 180 | 0 0 0 | 0 3 3 | 1 0 0 | 0 | 0 | 0 | 0 | 0 |
| F 3F519B10 | 87542073 | F 0 1 | — | 2 | 0 | 3 | FB | 1 1 1 | 0 | 180 | 0 0 0 | 0 3 3 | 1 0 0 | 0 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 1 1 | 0 | 1C0 | 0 0 0 | 0 3 3 | 1 0 0 | 0 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 1 1 | 0 | 180 | 0 0 0 | 0 3 3 | 1 0 0 | 0 | 0 | 0 | 0 | 0 |
| F 3F769689 | 5FFC05A0 | F 0 1 | — | 2 | 0 | 3 | FB | 1 1 1 | 0 | 180 | 0 0 0 | 0 3 3 | 1 0 0 | 0 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 1 1 | 0 | 1C0 | 0 0 0 | 0 3 3 | 1 0 0 | 0 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 1 1 | 0 | 180 | 0 0 0 | 0 3 3 | 1 0 0 | 0 | 0 | 0 | 0 | 0 |
| F 3FA375CA | 61D2988C | F 0 1 | — | 2 | 0 | 3 | FB | 1 1 1 | 0 | 180 | 0 0 0 | 0 3 3 | 1 0 0 | 0 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 1 1 | 0 | 1C0 | 0 0 0 | 0 3 3 | 1 0 0 | 0 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 1 1 | 0 | 180 | 0 0 0 | 0 3 3 | 1 0 0 | 0 | 0 | 0 | 0 | 0 |
| F 3FE7B204 | 23D1D930 | F 0 1 | — | 2 | 0 | 3 | FB | 1 1 1 | 0 | 180 | 0 0 0 | 0 3 3 | 1 0 0 | 0 | 0 | 0 | 0 | 0 |
| F 3FE6A09E | 667F3BCD | F 0 1 | — | 2 | 1 | 3 | BF | 1 1 1 | 0 | 1C0 | 0 0 0 | 0 3 3 | 1 0 0 | 0 | 0 | 0 | 0 | 0 |
| F 3FF6A09E | 667F3BCD | F 0 0 | — | 2 | 0 | 3 | EF | 1 1 1 | 0 | 1C0 | 0 0 0 | 0 3 3 | 1 0 0 | 0 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 1 0 | — | 2 | 0 | 3 | EF | 1 1 1 | 0 | 1C0 | 0 0 0 | 0 3 3 | 1 0 0 | 0 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FF | 1 1 1 | 0 | 180 | 0 0 0 | 0 3 3 | 1 0 0 | 0 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FF | 1 1 1 | 0 | 300 | 0 0 0 | 0 3 3 | 1 0 0 | 0 | 0 | 0 | 0 | 0 |
| F 3FF921FB | 54442D18 | F 0 1 | — | 2 | 1 | 3 | FB | 1 1 1 | 0 | 183 | 0 0 0 | 0 3 3 | 0 0 0 | 0 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FF | 1 1 1 | 0 | 300 | 0 0 0 | 0 3 3 | 1 0 0 | 0 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FF | 1 1 1 | 0 | 300 | 0 0 0 | 0 3 3 | 0 0 0 | 0 | 0 | 0 | 0 | 0 |

STEP 1 — Preprocessing: If the magnitude of the input X is greater than 1.0, the input is in range.

Steps required to perform the Calculation

All floating-point inputs are outputs are double precision. The output is in radians.

ArcTangent Routine Using CORDIC's Method

ArcTangent Routine Using Chebyshev's Method

All floating-point inputs and outputs are double precision. The output is in radians.

Steps Required to Perform the Calculation

STEP 1 — Preprocessing; If the magnitude of the input, X , is greater than 1.0, then the reciprocal must be taken. If the magnitude of X is not greater than 1.0, then pass X . Let this number (either X or $1.0/X$) be referred to as $X1$. Next multiply $X1$ times 2.0, then multiply this resulting number by $X1$. Finally, subtract 1.0 from this last product.

If $|X| > 1.0$

Then $X1 \leftarrow 1.0/X$

Else $X1 \leftarrow X$

$X2 \leftarrow X1 * 2.0 * X1 - 1.0$

STEP 2 — Core Calculation; $X2$ in Step 1 will be referred to as ' x ' in the core calculation.

$X3 \leftarrow C_{series_atan}$

$\leftarrow (((((((((((((((c19*x + c18)*x + c17)*x + c16)*x + c15)*x + c14)*x + c13)*x + c12)*x + c11)*x + c10)*x + c9)*x + c8)*x + c7)*x + c6)*x + c5)*x + c4)*x + c3)*x + c2)*x + c1)*x + c0$

STEP 3 — Postprocessing; multiply the output of the core calculation times $X1$. Let this number be referred to as $X4$. The next computation will yield the answer. If X was greater than 1.0, then subtract $X4$ from $\pi/2$. If X was less than -1.0 , then subtract $X4$ from $-\pi/2$. If neither of the two conditions above are true, then $X4$ is the answer.

$X4 \leftarrow X3 * X1$

If $X > 1.0$

Then $\text{ArcTangent}(X) \leftarrow \pi/2 - X4$

Else If $X < -1.0$

Then $\text{ArcTangent}(X) \leftarrow -\pi/2 - X4$

Else $\text{ArcTangent}(X) \leftarrow X4$

Algorithms for the Three Steps

Step 1 perform the preprocessing:

If $|X| > 1.0$

Then $T1 \leftarrow 1.0/X$

$T2 \leftarrow T1 * 2.0$

$T3 \leftarrow T2 * CREG$

$T4 \leftarrow T3 - 1.0$

Else $T1 \leftarrow X$

$T2 \leftarrow T1 * 2.0$

$T3 \leftarrow T2 * T1$

$T4 \leftarrow T3 - 1.0$

T1 is X1 in Step 1, must be stored externally

$CREG \leftarrow T1$

Step 2 perform the core calculation:

$T5 \leftarrow c_{19} * CREG$

$T6 \leftarrow T5 + c_{18}$

$T7 \leftarrow T6 * CREG$

$T8 \leftarrow T7 + c_{17}$

$T9 \leftarrow T8 * CREG$

$T10 \leftarrow T9 + c_{16}$

$T11 \leftarrow T10 * CREG$

$T12 \leftarrow T11 + c_{15}$

$T13 \leftarrow T12 * CREG$

$T14 \leftarrow T13 + c_{14}$

$T15 \leftarrow T14 * CREG$

$T16 \leftarrow T15 + c_{13}$

$T17 \leftarrow T16 * CREG$

$T18 \leftarrow T17 + c_{12}$

$T19 \leftarrow T18 * CREG$

$T20 \leftarrow T19 + c_{11}$

$T21 \leftarrow T20 * CREG$

$T22 \leftarrow T21 + c_{10}$

$T23 \leftarrow T22 * CREG$

$T24 \leftarrow T23 + c_9$

$T25 \leftarrow T24 * CREG$

$T26 \leftarrow T25 + c_8$

$T27 \leftarrow T26 * CREG$

$T28 \leftarrow T27 + c_7$

$T29 \leftarrow T28 * CREG$

$T30 \leftarrow T29 + c_6$

$CREG \leftarrow T4$

```

T31 ← T30 * CREG
T32 ← T31 + c5
T33 ← T32 * CREG
T34 ← T33 + c4
T35 ← T34 * CREG
T36 ← T35 + c3
T37 ← T36 * CREG
T38 ← T37 + c2
T39 ← T38 * CREG
T40 ← T39 + c1
T41 ← T40 * CREG
T42 ← T41 + c0

```

Step 3 perform the postprocessing:

```

T43 ← T42 * T1
If X > 1.0
    Then ArcTangent(X) ← pi/2 - CREG
    Return
If X < -1.0
    Then ArcTangent(X) ← -pi/2 - CREG
    Return
ArcTangent(X) ← CREG

```

Required System Intervention

As seen in the algorithm for Step 1, the 'ACT8847 performs a compare. The results of this compare determine what kind of preprocessing is to be performed. In Step 3, there are two more compare operations. The system must therefore perform additional decision making. In addition, the system must store T1, and later (in the postprocessing) provide this value to the 'ACT8847.

Number of 'ACT8847 Cycles Required to Calculate ArcTangent(x)

Calculation of ArcTangent(x) requires at most 89 cycles (including the divide instruction). In addition, it is assumed that 15 additional cycles are required due to the compare instructions, and resulting system intervention. Therefore, the total number of cycles to perform the ArcTangent(x) calculation is 104.

Listing of the Chebyshev Constants (c's)

The constants are represented in IEEE double-precision floating-point format.

| | | |
|-----|---|------------------|
| c19 | = | BDC4D6CC6308553F |
| c18 | = | 3DDFFD56FCFD2315 |
| c17 | = | BDE880782D99D071 |
| c16 | = | 3E0409670CB71218 |
| c15 | = | BE237C8239249B77 |
| c14 | = | 3E3F1358EC1D6AC0 |
| c13 | = | BE587CD25F4AFBED |
| c12 | = | 3E73D2388B0B8A86 |
| c11 | = | BE9028E921CA6A94 |
| c10 | = | 3EAA814997A38D4E |
| c9 | = | BEC5EDAD9A21FE5F |
| c8 | = | 3EE256E57BA07FAE |
| c7 | = | BEFF171F48FDF707 |
| c6 | = | 3F1ACFA9F95CA0DF |
| c5 | = | BF37A8464221D994 |
| c4 | = | 3F558DF7A83283C9 |
| c3 | = | BF749B3E2E433683 |
| c2 | = | 3F955A300BFB8078 |
| c1 | = | BFBA1494C19FADD4 |
| c0 | = | 3FEBDA7A85BD40CB |

Table 1.1. Legend code for Chebyshev polynomial (bits 0 - 010) and 1 - 00)

Legend code 1 bits for the Chebyshev(x) calculation

Pseudocode Table for the ArcTangent(x) Calculation

Table 11. Pseudocode for Chebyshev ArcTangent Routine (PIPES2-0 = 010, RND1-0 = 00)

| CLK | DA BUS | DB BUS | RA REG | RB REG | CLK MODE | INSTR | MUL PIPE | ALU PIPE | P REG | C REG | S REG | Y BUS | COMMENT |
|-----|-----------|-----------|-----------|-----------|-------------|----------------------|-------------|-------------|----------|----------|----------|----------|---|
| 1 | 1.0 MSH | 1.0 LSH | 0 | | | COMPARE RA2, RB2 | | | | | | | X is the input Compare 1.0 and ABS(X) |
| 2 | X MSH | X LSH | 1.0 | X | 0 | RA2*RB2 RA2, RB2 | | | | | | | If ABS(X) is greater than 1.0 perform 1.0/X, other- wise go to cycle 16b |
| 3 | | | 1.0 | X | 0 | NOP | | | | | | | Wait for system response |
| 4 | | | 1.0 | X | 1 | DIV | | | | | | | Divide: 1.0/X |
| 5 | | | 1.0 | X | 0 | NOP | | | | | | | Wait for Division result |
| 6 | | | 1.0 | X | 0 | NOP | | | | | | | Wait for Division result |
| 7 | | | 1.0 | X | 0 | NOP | | | | | | | Wait for Division result |
| 8 | | | 1.0 | X | 0 | NOP | | | | | | | Wait for Division result |
| 9 | | | 1.0 | X | 0 | NOP | | | | | | | Wait for Division result |
| 10 | | | 1.0 | X | 0 | NOP | | | | | | | Wait for Division result |
| 11 | | | 1.0 | X | 0 | NOP | | | | | | | Wait for Division result |
| 12 | | | 1.0 | X | 0 | NOP | | | | | | | Wait for Division result |
| 13 | | | 1.0 | X | 0 | NOP | | | | | | | Wait for Division result |
| 14 | | | 1.0 | X | 0 | NOP | | | | | | | Wait for Division result |
| 15 | | | 1.0 | X | 0 | NOP | | | | | | | Wait for Division result |
| 16a | 2.0 MSH | 2.0 LSH | 1.0 | X | 0 | RA17*PR17 | | | P1 | | | P1 | If the reciprocal of X was performed, then execute cycles 16a through 19a |
| 17a | | | 2.0 | X | 0 | RA17*PR17 | | | | | | P1 | |
| 18a | | | 2.0 | X | 0 | CR19*PR19 | RA17*PR17 | | | | P1 | | |
| 19a | | | 2.0 | X | 0 | CR19*PR19 | | | P2a | | | | In cycles 16a and 17 a out- put P1 and store it for use in cycle 79 |
| 16b | 2.0 MSH | 2.0 LSH | 1.0 | X | 0 | RA17*RB17 | | | | | | | If the reciprocal of X was |

Table 11. Pseudocode for Chebyshev ArcTangent Routine (PIPES2-0 = 010, RND1-0 = 00) (Continued)

| CLK | DA BUS | DB BUS | RA REG | RB REG | CLK MODE | INSTR | MUL PIPE | ALU PIPE | P REG | C REG | S REG | Y BUS | COMMENT |
|-----|---------------------|---------------------|----------------|-----------------|-------------|------------------------------|------------------------------|-------------|------------------|----------|----------|----------|--|
| 17b | | | 2.0 | X | 0 | RA17*RB17 | | | | | | | not performed, then execute |
| 18b | X MSH | X LSH | X | X | 0 | RA19*PR19 | RA17*RB17 | | | | | | cycle 16b through 19b |
| 19b | | | X | X | 0 | RA19*PR19 | | | P2b | | | | |
| 20 | | | 2.0 or X | X | 0 | PR21+RB21 or RA19*PR19 | CR19*PR19 or RA19*PR19 | | | | | | The RA register is not used again until cycle 81 so rather than indicating |
| 21 | -1.0 MSH | -1.0 LSH | 2.0 or X | -1.0 | 0 | PR21+RB21 | | | P3a or P3b | | | | the contents ' 2.0 of RA as: or X ' |
| 22 | c ₁₉ MSH | c ₁₉ LSH | 2 or X | c ₁₉ | 1 | SR22*RB22 | | | | | S1 | | use the term ' 2 or X' Start the core calculation |
| 23 | | | 2 or X | c ₁₉ | 0 | PR24+RB24 | SR22*RB22 | | | S1 | | | |
| 24 | c ₁₈ MSH | c ₁₈ LSH | 2 or X | c ₁₈ | 0 | PR24+RB24 | | | P4 | | | | |
| 25 | | | 2 or X | c ₁₈ | 1 | SR25*CR25 | | | | | S2 | | |
| 26 | | | 2 or X | c ₁₈ | 0 | PR27+RB27 | SR25*CR25 | | | | | | |
| 27 | c ₁₇ MSH | c ₁₇ LSH | 2 or X | c ₁₇ | 0 | PR27+RB27 | | | P5 | | | | |
| 28 | | | 2 or X | c ₁₇ | 1 | SR28*CR28 | | | | | S3 | | |
| 29 | | | 2 or X | c ₁₇ | 0 | PR30+RB30 | SR28*CR28 | | | | | | |
| 30 | c ₁₆ MSH | c ₁₆ LSH | 2 or X | c ₁₆ | 0 | PR30+RB30 | | | P6 | | | | |
| 31 | | | 2 or X | c ₁₆ | 1 | SR31*CR31 | | | | | S4 | | |
| 32 | | | 2 or X | c ₁₆ | 0 | PR33+RB33 | SR31*CR31 | | | | | | |
| 33 | c ₁₅ MSH | c ₁₅ LSH | 2 or X | c ₁₅ | 0 | PR33+RB33 | | | P7 | | | | |
| 34 | | | 2 or X | c ₁₅ | 1 | SR34*CR34 | | | | | S5 | | |
| 35 | | | 2 or X | c ₁₅ | 0 | PR36+RB36 | SR34*CR34 | | | | | | |
| 36 | c ₁₄ MSH | c ₁₄ LSH | 2 or X | c ₁₄ | 0 | PR36+RB36 | | | P8 | | | | |

Table 11. Pseudocode for Chebyshev ArcTangent Routine (PIPES2-0 = 010, RND1-0 = 00) (Continued)

| CLK | DA BUS | DB BUS | RA REG | RB REG | CLK MODE | INSTR | MUL PIPE | ALU PIPE | P REG | C REG | S REG | Y BUS | COMMENT |
|-----|-----------|-----------|-----------|-----------|-------------|-----------|-------------|-------------|----------|----------|----------|----------|---------|
| 37 | | | 2 or X | c14 | 1 | SR37*CR37 | | | | | S6 | | |
| 38 | | | 2 or X | c14 | 0 | PR39+RB39 | SR37*CR37 | | | | | | |
| 39 | c13 MSH | c13 LSH | 2 or X | c13 | 0 | PR39+RB39 | | | P9 | | | | |
| 40 | | | 2 or X | c13 | 1 | SR40*CR40 | | | | | S7 | | |
| 41 | | | 2 or X | c13 | 0 | PR42+RB42 | SR40*CR40 | | | | | | |
| 42 | c12 MSH | c12 LSH | 2 or X | c12 | 0 | PR42+RB42 | | | P10 | | | | |
| 43 | | | 2 or X | c12 | 1 | SR43*CR43 | | | | | S8 | | |
| 44 | | | 2 or X | c12 | 0 | PR45+RB45 | SR43*CR43 | | | | | | |
| 45 | c11 MSH | c11 LSH | 2 or X | c11 | 0 | PR45+RB45 | | | P11 | | | | |
| 46 | | | 2 or X | c11 | 1 | SR46*CR46 | | | | | S9 | | |
| 47 | | | 2 or X | c11 | 0 | PR48+RB48 | SR46*CR46 | | | | | | |
| 48 | c10 MSH | c10 LSH | 2 or X | c10 | 0 | PR48+RB48 | | | P12 | | | | |
| 49 | | | 2 or X | c10 | 1 | SR49*CR49 | | | | | S10 | | |
| 50 | | | 2 or X | c10 | 0 | PR51+RB51 | SR49*CR49 | | | | | | |
| 51 | c9 MSH | c9 LSH | 2 or X | c9 | 0 | PR51+RB51 | | | P13 | | | | |
| 52 | | | 2 or X | c9 | 1 | SR52*CR52 | | | | | S11 | | |
| 53 | | | 2 or X | c9 | 0 | PR54+RB54 | SR52*CR52 | | | | | | |
| 54 | c8 MSH | c8 LSH | 2 or X | c8 | 0 | PR54+RB54 | | | P14 | | | | |
| 55 | | | 2 or X | c8 | 1 | SR55*CR55 | | | | | S12 | | |
| 56 | | | 2 or X | c8 | 0 | PR57+RB57 | SR55*CR55 | | | | | | |
| 57 | c7 MSH | c7 LSH | 2 or X | c7 | 0 | PR57+RB57 | | | P15 | | | | |
| 58 | | | 2 or X | c7 | 1 | SR58*CR58 | | | | | S13 | | |
| 59 | | | 2 or X | c7 | 0 | PR60+RB60 | SR58*CR58 | | | | | | |
| 60 | c6 MSH | c6 LSH | 2 or X | c6 | 0 | PR60+RB60 | | | P16 | | | | |

Table 11. Pseudocode for Chebyshev ArcTangent Routine (PIPES2-0 = 010, RND1-0 = 00) (Continued)

| CLK | DA BUS | DB BUS | RA REG | RB REG | CLK MODE | INSTR | MUL PIPE | ALU PIPE | P REG | C REG | S REG | Y BUS | COMMENT |
|-----|--------------------|--------------------|-----------|----------------|-------------|------------------|-------------|-------------|----------|----------|----------|----------|---|
| 61 | | | 2 or X | c ₆ | 1 | SR61•CR61 | | | | | S14 | | |
| 62 | | | 2 or X | c ₆ | 0 | PR63+RB63 | SR61•CR61 | | | | | | |
| 63 | c ₅ MSH | c ₅ LSH | 2 or X | c ₅ | 0 | PR63+RB63 | | | P17 | | | | |
| 64 | | | 2 or X | c ₅ | 1 | SR64•CR64 | | | | | S15 | | |
| 65 | | | 2 or X | c ₅ | 0 | PR66+RB66 | SR64•CR64 | | | | | | |
| 66 | c ₄ MSH | c ₄ LSH | 2 or X | c ₄ | 0 | PR66+RB66 | | | P18 | | | | |
| 67 | | | 2 or X | c ₄ | 1 | SR67•CR67 | | | | | S16 | | |
| 68 | | | 2 or X | c ₄ | 0 | PR69+RB69 | SR67•CR67 | | | | | | |
| 69 | c ₃ MSH | c ₃ LSH | 2 or X | c ₃ | 0 | PR69+RB69 | | | P19 | | | | |
| 70 | | | 2 or X | c ₃ | 1 | SR70•CR70 | | | | | S17 | | |
| 71 | | | 2 or X | c ₃ | 0 | PR72+RB72 | SR70•CR70 | | | | | | |
| 72 | c ₂ MSH | c ₂ LSH | 2 or X | c ₂ | 0 | PR72+RB72 | | | P20 | | | | |
| 73 | | | 2 or X | c ₂ | 1 | SR73•CR73 | | | | | S18 | | |
| 74 | | | 2 or X | c ₂ | 0 | PR75+RB75 | SR73•CR73 | | | | | | |
| 75 | c ₁ MSH | c ₁ LSH | 2 or X | c ₁ | 0 | PR75+RB75 | | | P21 | | | | |
| 76 | | | 2 or X | c ₁ | 1 | SR76•CR76 | | | | | S19 | | |
| 77 | | | 2 or X | c ₁ | 0 | PR78+RB78 | SR76•CR76 | | | | | | |
| 78 | c ₀ MSH | c ₀ LSH | 2 or X | c ₀ | 0 | PR78+RB78 | | | P22 | | | | |
| 79 | T1 MSH | T1 LSH | 2 or X | T1 | 1 | SR79•RB79 | | | | | S20 | | T1 is either P1 or is X depending on what action was called for at cycle 2 Begin the post processing |
| 80 | X MSH | X LSH | 2 or X | T1 | 0 | COMPARE X,1.0 | | | | | | | If X > 1.0 then execute 83 through 86, otherwise skip to 83b. In either case execute 80 through 82 |

Table 11. Pseudocode for Chebyshev ArcTangent Routine (PIPES2-0 = 010, RND1-0 = 00) (Concluded)

| CLK | DA BUS | DB BUS | RA REG | RB REG | CLK MODE | INSTR | MUL PIPE | ALU PIPE | P REG | C REG | S REG | Y BUS | COMMENT |
|-----|--------------|--------------|-----------|-----------|-------------|-------------------|-------------|-------------|----------|----------|----------|----------|--|
| 81 | 1.0 MSH | 1.0 LSH | X | 1.0 | 0 | COMPARE X,1.0 | | | P23 | | | | |
| 82 | | | X | 1.0 | 0 | NOP | | | | P23 | | | Wait for system response |
| 83 | | | X | 1.0 | 0 | RB84 - CR84 | | | | | | | Execute if X > 1.0 |
| 84 | pi/2 MSH | pi/2 LSH | X | pi/2 | 0 | RB84 - CR84 | | | | | | | |
| 85 | | | X | pi/2 | 0 | NOP | | | | | S21a | S21a | Output MSH of answer |
| 86 | | | X | pi/2 | 0 | NOP | | | | | S21a | S21a | Output LSH of answer The calculation is done |
| 83b | -1.0 MSH | -1.0 LSH | X | 1.0 | 0 | COMPARE -1.0,X | | | | | | | Execute if X ≤ 1.0. If -1.0 > X then execute 86b through 89b, otherwise skip to 86c. In either case execute 83b thru 85b |
| 84b | X MSH | X LSH | -1.0 | X | 0 | COMPARE -1.0,X | | | | P23 | | | |
| 85b | | | -1.0 | X | 0 | NOP | | | | P23 | | | Wait for system response |
| 86b | | | -1.0 | X | 0 | RB87 - CR87 | | | | | | | Execute if -1.0 > X |
| 87b | -pi/2 MSH | -pi/2 LSH | -1.0 | -pi/2 | 0 | RB87 - CR87 | | | | | | | |
| 88b | | | -1.0 | -pi/2 | 0 | NOP | | | | | S21b | S21b | Output MSH of answer |
| 89b | | | -1.0 | pi/2 | 0 | NOP | | | | | S21b | S21b | Output LSH of answer. The calculation is done. |
| 86c | | | -1.0 | X | 1 | PASS(CR86) | | | | | | | Execute if X is within the range [-1,1], Pass CREG |
| 87c | | | -1.0 | X | 0 | NOP | | | | | S21c | S21c | Output MSH of answer |
| 88c | | | -1.0 | X | 0 | NOP | | | | | S21c | S21c | Output LSH of answer |

Table 11. Pseudocode for Chebyshev ArcTangent Routine (PIPES2-0 = 010, RND1-0 = 00) (Continued)

Microcode Table for the ArcTangent(x) Calculation

All numbers are in hex. Any field with a length that is not a multiple of 4 is right justified and zero filled. In the table, the value of X has been chosen to be SQRT(3.0).

[illegible]

Microcode Table for the ArcTangent(x) Calculation (Continued)

| P A | D A | D B | P B | E N A | E N B | C L K C | P I P E S | C L O M F O I D G | S E L O P E T | R E S E T | H A N C | E L C | F L O W C | I N S T R | R N D | F N A R S C T | S A R Y T C E P T | B Y T E S T E S T Y | S E E L Y | S E E L Y | O E S C | O E S C | O E S C | |
|------------|----------|--------|--------|-------------|-------------|------------------|-----------------------|---|---------------------------------|-----------------------|------------------|-------------|-----------------------|-----------------------|-------------|---------------------------------|---|--|-----------------------|-----------------------|------------------|------------------|------------------|--|
| F 00000000 | 00000000 | F 0 0 | ┐ | 2 | 0 | 3 | FB | 1 | 1 | 0 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | |
| F 3DDFFD56 | FCFD2315 | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | |
| F BDE88078 | 2D99D071 | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | |
| F 3E040967 | 0CB71218 | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | |
| F BE237C82 | 39249B77 | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | |
| F 3E3F1358 | EC1D6AC0 | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | |
| F BE587CD2 | 5F4AFBED | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | |
| F 3E73D238 | 8B0B8A86 | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | |
| F BE9028E9 | 21CA6A94 | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | |

Microcode Table for the ArcTangent(x) Calculation (Continued)

| P | D | D | P | E | C | P | C | S | R | H | E | F | I | R | F | S | B | S | T | S | O | O | O |
|------------|----------|-------|---|---|---|---|----|---|---|---|---|-----|---|---|---|---|---|---|---|---|---|---|---|
| A | A | B | B | N | N | L | I | L | O | E | E | A | N | L | N | N | A | R | Y | E | E | E | E |
| | | | A | B | K | P | K | N | L | S | L | C | O | S | D | S | C | T | L | S | L | Y | S |
| | | | | | C | E | M | F | O | E | T | W | T | | T | C | E | S | T | Y | | | |
| | | | | | | S | O | I | P | T | | C | R | | | | P | T | | | | | |
| | | | | | | | | | D | G | | | | | | | | | | | | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 |
| F 3EAA8149 | 97A38D4E | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 |
| F BEC5EDAD | 9A21FE5F | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 |
| F 3EE256E5 | 7BA07FAE | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 |
| F BEFF171F | 48FDF707 | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 |
| F 3F1ACFA9 | F95CA0DF | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 |
| F BF37A846 | 4221D994 | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 |
| F 3F558DF7 | A83283C9 | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 |
| F BF749B3E | 2E433683 | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 |

Microcode Table for the ArcTangent(x) Calculation (Concluded)

| P | D | P | E | E | C | P | C | C | S | R | H | E | F | I | R | F | S | B | S | T | S | O | O | O |
|------------|----------|-------|---|---|---|---|----|---|---|---|---|-----|---|---|---|---|---|---|---|---|---|---|---|---|
| A | A | B | N | N | L | I | L | O | E | E | A | N | L | N | N | A | R | Y | E | E | E | E | E | E |
| | | | A | B | K | P | K | N | L | S | L | C | O | S | D | S | C | T | L | S | L | Y | S | C |
| | | | | | C | E | M | F | O | E | T | | W | T | | T | C | E | S | T | Y | | | |
| | | | | | S | O | I | P | T | | | C | R | | | | P | T | | | | | | |
| | | | | | D | G | | | | | | | | | | | | | | | | | | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 | |
| F 3F955A30 | 0BFB8078 | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 | |
| F BFBA1494 | C19FADD4 | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 | |
| F 3FEBDA7A | 85BD40CB | F 0 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 | |
| F 3FE279A7 | 4590331C | F 0 1 | — | 2 | 1 | 3 | BF | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 | |
| F 3FFBB67A | E8584CAB | F 0 0 | — | 2 | 0 | 3 | FF | 1 | 1 | 1 | 0 | 182 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 | |
| F 3FF00000 | 00000000 | F 1 1 | — | 2 | 0 | 3 | FF | 1 | 1 | 1 | 0 | 182 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 | |
| F 00000000 | 00000000 | F 0 0 | ┘ | 2 | 0 | 3 | FF | 1 | 1 | 0 | 0 | 300 | 0 | 0 | 1 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | F7 | 1 | 1 | 1 | 0 | 183 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 | |
| F 3FF921FB | 54442D18 | F 0 1 | — | 2 | 0 | 3 | F7 | 1 | 1 | 1 | 0 | 183 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FF | 1 | 1 | 1 | 0 | 300 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | 0 | |
| F 00000000 | 00000000 | F 0 0 | — | 2 | 0 | 3 | FF | 1 | 1 | 1 | 0 | 300 | 0 | 0 | 0 | 0 | 3 | 3 | 0 | 0 | 0 | 0 | 0 | |

Microcode Table for the ArcTangent(x) Calculation (Continued)

Exponential Routine Using Chebyshev's Method

All floating-point inputs and outputs are double precision.

Steps Required to Perform the Calculation

STEP 1 — Preprocessing; first multiply the input, X , by $\log_2 e$ (yielding $X1$). Next, convert this product to an integer, using truncate mode (yielding $X2$). Form the variable EX by adding 1024 to $X2$. EX is used in the postprocessing part of the routine. Subtract 1023 from EX to find the variable N (N is actually $X2$ incremented by 1). Convert N to a floating-point number (yielding $X3$). Subtract $X1$ from $X3$, multiply this difference by 2.0, and then finally subtract 1.0. This last computation is the input to the core routine.

```
X1 ← X*log2e
X2 ← TRUNC(X1)
EX ← 1024 + X2
N ← EX - 1023
X3 ← DOUBLE(N)
X4 ← 2.0*(X3 - X1) - 1.0
```

STEP 2 — Core Calculation; $X4$ in Step 1 will be referred to as ' x ' in the core calculation.

```
X5 ← Cseries_exp
← (((((((((c11*x + c10)*x + c9)*x + c8)*x + c7)*x + c6)*x +
c5)*x + c4)*x + c3)*x + c2)*x + c1)*x + c0
```

STEP 3 — Postprocessing; multiply the output of the core calculation times 2^N . To generate 2^N , perform the following: shift left logical 20 positions (bits) the variable EX (which was calculated in Step 1). The resulting bit pattern will be the double precision floating-point representation of 2^N . However, the 'ACT8847 will not at this point recognize the bit pattern as a floating-point number. So this number must be output from the Y bus, and then input (declaring the input to be a double precision floating-point number) on the input bus. Now the 'ACT8847 will process 2^N as a double float, and so the core output, $X5$, can be multiplied by 2^N to produce the final result. 'SLL' means to shift left logical.

```
X6 ← EX SLL by 20 bits
Y bus ← X6
DA bus ← Y bus
Exp(X) ← X5 * X6
```

Algorithms for the Three Steps

Step 1 perform the preprocessing:

| | |
|-----------------------------------|---|
| T1 $\leftarrow X \cdot \log_2 e$ | $\log_2 e$ entered as a constant |
| T2 $\leftarrow \text{INT}(T1)$ | round controls set to truncate |
| T3 $\leftarrow 1024 + T2$ | T3 is EX in Step 1, must be stored externally, CREG $\leftarrow T1$ |
| T4 $\leftarrow T3 - 1023$ | |
| T5 $\leftarrow 1 \cdot T4$ | makes T4 available to A2 MUX |
| T6 $\leftarrow \text{DOUBLE}(T5)$ | convert from integer to double |
| T7 $\leftarrow T6 - \text{CREG}$ | |
| T8 $\leftarrow 2.0 \cdot T7$ | |
| T9 $\leftarrow T8 - 1.0$ | T9 is X4 in Step 1, the input to the core routine |

Step 2 perform the core calculation:

| | |
|---|----------------------|
| T10 $\leftarrow c_{11} \cdot \text{CREG}$ | |
| T11 $\leftarrow T10 + c_{10}$ | CREG $\leftarrow T9$ |
| T12 $\leftarrow T11 \cdot \text{CREG}$ | |
| T13 $\leftarrow T12 + c_9$ | |
| T14 $\leftarrow T13 \cdot \text{CREG}$ | |
| T15 $\leftarrow T14 + c_8$ | |
| T16 $\leftarrow T15 \cdot \text{CREG}$ | |
| T17 $\leftarrow T16 + c_7$ | |
| T18 $\leftarrow T17 \cdot \text{CREG}$ | |
| T19 $\leftarrow T18 + c_6$ | |
| T20 $\leftarrow T19 \cdot \text{CREG}$ | |
| T21 $\leftarrow T20 + c_5$ | |
| T22 $\leftarrow T21 \cdot \text{CREG}$ | |
| T23 $\leftarrow T22 + c_4$ | |
| T24 $\leftarrow T23 \cdot \text{CREG}$ | |
| T25 $\leftarrow T24 + c_3$ | |
| T26 $\leftarrow T25 \cdot \text{CREG}$ | |
| T27 $\leftarrow T26 + c_2$ | |
| T28 $\leftarrow T27 \cdot \text{CREG}$ | |
| T29 $\leftarrow T28 + c_1$ | |
| T30 $\leftarrow T29 \cdot \text{CREG}$ | |
| T31 $\leftarrow T30 + c_0$ | |

Step 3 perform the postprocessing:

T32 \leftarrow T3 SLL by 20 bits
Y bus \leftarrow T32

DA bus \leftarrow Y bus (= T32)

Exp(X) \leftarrow T32*CREG

Shift T3 20 bits left
Output and then Input T32
CREG \leftarrow T31
Two cycles required to
input both halves of T32

Required System Intervention

The system is required to store the variable EX, and then later provide this variable. In addition, the system is required to route the variable T32 (in Step 3) from the Y bus to the DA bus.

Number of 'ACT8847 Cycles Required to Calculate Exp(x)

Calculation of Exp(x) requires 52 cycles. Since there are no decisions which the system is required to perform, the total number of cycle to perform the Exp(X) calculation is 52.

Listing of the Chebyshev Constants (c's)

The constants are represented in IEEE double-precision floating-point format.

c11 = BD45A7FC05D3B501
c10 = 3D957BFD2DBF487C
c9 = BDE351B821AC16D5
c8 = 3E2F5B0E17440879
c7 = BE769E51EE631E87
c6 = 3EBC8D7530548DD5
c5 = BEFEE4FD234A4926
c4 = 3F3BDB696E8987AC
c3 = BF741839EB88156E
c2 = 3FA5BE298ADF0369
c1 = BFCE5E46537AB906
c0 = 3FE6A09E667F3BCC

Pseudocode Table for the Exp(x) Calculation

Table 12. Pseudocode for Chebyshev Exponential Routine (PIPES2-0 = 010, RND1-0)

| CLK | DA BUS | DB BUS | RA REG | RB REG | CLK MODE | INSTR | MUL PIPE | ALU PIPE | P REG | C REG | S REG | Y BUS | COMMENT |
|-----|---------------------------|---------------------------|-----------|--------------------|-------------|-------------|-------------|-------------|----------|----------|----------|----------|---|
| 1 | X MSH | X LSH | | | 0 | RA2*RB2 | | | | | | | X is the input |
| 2 | Log ₂ e MSH | Log ₂ e LSH | X | Log ₂ e | 0 | RA2*RB2 | | | | | | | |
| 3 | | | X | Log ₂ e | 0 | DP2I(PR4) | RA2*RB2 | | | | | | Double-precision → integer |
| 4 | | | X | Log ₂ e | 0 | DP2I(PR4) | | | P1 | | | | |
| 5 | 1024 | | 1024 | Log ₂ e | 0 | RA5 + SR5 | | | | P1 | S1 | | |
| 6 | -1023 | | -1023 | Log ₂ e | 0 | RA6 + SR6 | | | | | S2 | S2 | Store S2, which is the variable EX, for use in cycle 46 |
| 7 | | 1 | -1023 | 1 | 0 | SR7*RB7 | | | | | S3 | | |
| 8 | | | -1023 | 1 | 1 | I2DP(PR8) | | | P2 | | | | Integer → double-precision |
| 9 | | | -1023 | 1 | 1 | SR9 - CR9 | | | | | S4 | | |
| 10 | 2.0 MSH | 2.0 LSH | -1023 | 2.0 | 1 | SR10*RB10 | | | | | S5 | | |
| 11 | | | -1023 | 2.0 | 0 | PR12 + RB12 | SR10*RB10 | | | | | | |
| 12 | -1.0 MSH | -1.0 LSH | -1023 | -1.0 | 0 | PR12 + RB12 | | | P3 | | | | |
| 13 | c ₁₁ MSH | c ₁₁ LSH | -1023 | c ₁₁ | 1 | SR13*RB13 | | | | | S6 | | Start core calculation, S6 is the input to the core calculation |
| 14 | | | -1023 | c ₁₁ | 0 | PR15 + RB15 | SR13*RB13 | | | | S6 | | |
| 15 | c ₁₀ MSH | c ₁₀ LSH | -1023 | c ₁₀ | 0 | PR15 + RB15 | | | P4 | | | | |
| 16 | | | -1023 | c ₁₀ | 1 | SR16*CR16 | | | | | S7 | | |
| 17 | | | -1023 | c ₁₀ | 0 | PR18 + RB18 | SR16*CR16 | | | | | | |
| 18 | c ₉ MSH | c ₉ LSH | -1023 | c ₉ | 0 | PR18 + RB18 | | | P5 | | | | |
| 19 | | | -1023 | c ₉ | 1 | SR19*CR19 | | | | | S8 | | |
| 20 | | | -1023 | c ₉ | 0 | PR21 + RB21 | SR19*CR19 | | | | | | |

Table 12. Pseudocode for Chebyshev Exponential Routine (PIPES2-0 = 010, RND1-0) (Continued)

| CLK | DA BUS | DB BUS | RA REG | RB REG | CLK MODE | INSTR | MUL PIPE | ALU PIPE | P REG | C REG | S REG | Y BUS | COMMENT |
|-----|--------------------|--------------------|-----------|----------------|-------------|-------------|-------------|-------------|----------|----------|----------|----------|---------|
| 21 | c ₈ MSH | c ₈ LSH | -1023 | c ₈ | 0 | PR21 + RB21 | | | P6 | | | | |
| 22 | | | -1023 | c ₈ | 1 | SR22 • CR22 | | | | | S9 | | |
| 23 | | | -1023 | c ₈ | 0 | PR24 + RB24 | SR22 • CR22 | | | | | | |
| 24 | c ₇ MSH | c ₇ LSH | -1023 | c ₇ | 0 | PR24 + RB24 | | | P7 | | | | |
| 25 | | | -1023 | c ₇ | 1 | SR25 • CR25 | | | | | S10 | | |
| 26 | | | -1023 | c ₇ | 0 | PR27 + RB27 | SR25 • CR25 | | | | | | |
| 27 | c ₆ MSH | c ₆ LSH | -1023 | c ₆ | 0 | PR27 + RB27 | | | P8 | | | | |
| 28 | | | -1023 | c ₆ | 1 | SR28 • CR28 | | | | | S11 | | |
| 29 | | | -1023 | c ₆ | 0 | PR30 + RB30 | SR28 • CR28 | | | | | | |
| 30 | c ₅ MSH | c ₅ LSH | -1023 | c ₅ | 0 | PR30 + RB30 | | | P9 | | | | |
| 31 | | | -1023 | c ₅ | 1 | SR31 • CR31 | | | | | S12 | | |
| 32 | | | -1023 | c ₅ | 0 | PR33 + RB33 | SR31 • CR31 | | | | | | |
| 33 | c ₄ MSH | c ₄ LSH | -1023 | c ₄ | 0 | PR33 + RB33 | | | P10 | | | | |
| 34 | | | -1023 | c ₄ | 1 | SR34 • CR34 | | | | | S13 | | |
| 35 | | | -1023 | c ₄ | 0 | PR36 + RB36 | SR34 • CR34 | | | | | | |
| 36 | c ₃ MSH | c ₃ LSH | -1023 | c ₃ | 0 | PR36 + RB36 | | | P11 | | | | |
| 37 | | | -1023 | c ₃ | 1 | SR37 • CR37 | | | | | S14 | | |
| 38 | | | -1023 | c ₃ | 0 | PR39 + RB39 | SR37 • CR37 | | | | | | |
| 39 | c ₂ MSH | c ₂ LSH | -1023 | c ₂ | 0 | PR39 + RB39 | | | P12 | | | | |
| 40 | | | -1023 | c ₂ | 1 | SR40 • CR40 | | | | | S15 | | |
| 41 | | | -1023 | c ₂ | 0 | PR42 + RB42 | SR40 • CR40 | | | | | | |
| 42 | c ₁ MSH | c ₁ LSH | -1023 | c ₁ | 0 | PR42 + RB42 | | | P13 | | | | |
| 43 | | | -1023 | c ₁ | 1 | SR43 • CR43 | | | | | S16 | | |

Table 12. Pseudocode for Chebyshev Exponential Routine (PIPES2-0 = 010, RND1-0) (Continued)

Table 12. Pseudocode for Chebyshev Exponential Routine (PIPES2-0 = 010, RND1-0) (Continued)

| CLK | DA BUS | DB BUS | RA REG | RB REG | CLK MODE | INSTR | MUL PIPE | ALU PIPE | P REG | C REG | S REG | Y BUS | COMMENT |
|-----|--------------------|--------------------|-----------|----------------|-------------|-------------------|-------------|-------------|----------|----------|----------|----------|--|
| 44 | | | -1023 | c ₁ | 0 | PR45 + RB45 | SR43 • CR43 | | | | | | |
| 45 | c ₀ MSH | c ₀ LSH | -1023 | c ₀ | 0 | PR45 + RB45 | | | P14 | | | | |
| 46 | S2 | 20 | S2 | 20 | 0 | SLL RA46, RB46 | | | | | | | Begin post processing. S2 is the variable EX, and was calculated in cycle 5. Shift left logical S2 20 bit positions |
| 47 | | | S2 | 20 | 0 | NOP | | | | S17 | S18 | S18 | Allows time for S18 to be output from the Y bus and input to the DA bus |
| 48 | S18 | | S2 | 20 | 0 | RA48 • CR48 | | | | | | | |
| 49 | 0 | | S18' | 20 | 0 | RA48 • CR48 | | | | | | | RA holds S18', which is the double precision floating-point equivalent of 2 ^N , where N was calculated in cycle 6 |
| 50 | | | S18' | 20 | 0 | DUMMY | RA48 • CR48 | | | | | | Instruction is RA + RB, used to allow time for result to propagate to Y bus |
| 51 | | | S18' | 20 | 0 | NOP | | | P15 | | | P15 | Output MSH of answer |
| 52 | | | S18' | 20 | 0 | NOP | | | P15 | | | P14 | Output LSH of answer |

Table 12. Pseudocode for Chebyshev Exponential Routine (PIPES2-0 = 010, RND1-0) (Continued)

Microcode Table for the Exp(x) Calculation

All numbers are in hex. Any field with a length that is not a multiple of 4 is right justified and zero filled. For the microcode table, the value of X has been chosen to be 6.25.

| P | D | D | P | E | E | C | P | C | C | S | R | H | E | F | I | R | F | S | B | S | T | S | O | O | O |
|------------|----------|-------|---|---|---|----|---|---|---|---|-----|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | A | B | B | N | N | L | I | L | O | E | E | A | N | L | N | N | A | R | Y | E | E | E | E | E | E |
| | | | | A | B | K | P | K | N | L | S | L | C | O | S | D | S | C | T | L | S | L | Y | S | C |
| | | | | | | C | E | M | F | O | E | T | | W | T | | T | C | E | S | T | Y | | | |
| | | | | | | | S | O | I | P | T | | | C | R | | | | P | T | | | | | |
| | | | | | | | D | G | | | | | | | | | | | | | | | | | |
| F 40190000 | 00000000 | F 0 0 | 2 | 0 | 3 | FF | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | | |
| F 3FF71547 | 652B82FE | F 1 1 | 2 | 0 | 3 | FF | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | | |
| F 00000000 | 00000000 | F 0 0 | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 1A3 | 1 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | | |
| F 00000000 | 00000000 | F 0 0 | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 1A3 | 1 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | | |
| F 00000400 | 00000000 | F 1 0 | 2 | 0 | 1 | FE | 1 | 1 | 0 | 0 | 200 | 0 | 0 | 1 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | | |
| F FFFFC01 | 00000000 | F 1 0 | 2 | 0 | 1 | FE | 1 | 1 | 1 | 0 | 200 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | | |
| F 00000000 | 00000001 | F 0 1 | 2 | 0 | 1 | BF | 1 | 1 | 1 | 0 | 240 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | | |
| F 00000000 | 00000000 | F 0 0 | 2 | 1 | 3 | FB | 1 | 1 | 1 | 0 | 1A2 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | | |
| F 00000000 | 00000000 | F 0 0 | 2 | 1 | 3 | F6 | 1 | 1 | 1 | 0 | 183 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | | |
| F 40000000 | 00000000 | F 0 1 | 2 | 1 | 3 | BF | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | | |
| F 00000000 | 00000000 | F 0 0 | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | | |
| F BFF00000 | 00000000 | F 0 1 | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | | |
| F BD45A7FC | 05D3B501 | F 0 1 | 2 | 1 | 3 | BF | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | | |
| F 00000000 | 00000000 | F 0 0 | 2 | 0 | 3 | FB | 1 | 1 | 0 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | | |
| F 3D957BFD | 2DBF487C | F 0 1 | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | | |
| F 00000000 | 00000000 | F 0 0 | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | | |
| F 00000000 | 00000000 | F 0 0 | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | | |
| F BDE351B8 | 21AC16D5 | F 0 1 | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | | |
| F 00000000 | 00000000 | F 0 0 | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | | |
| F 00000000 | 00000000 | F 0 0 | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | | |
| F 3E2F5B0E | 17440879 | F 0 1 | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 | | | | |

Microcode Table for the Exp(x) Calculation (Continued)

| P | D | D | P | E | E | C | P | C | C | S | R | H | E | F | I | R | F | S | B | S | T | S | O | O | O |
|---|----------|----------|---|---|---|---|---|---|---|----|---|---|---|---|-----|---|---|---|---|---|---|---|---|---|---|
| A | A | B | B | N | N | L | I | L | O | E | E | A | N | L | N | N | A | R | Y | E | E | E | E | E | E |
| | | | A | B | K | C | P | K | N | L | S | L | C | O | S | D | S | C | T | L | S | L | Y | S | C |
| | | | | | | | E | M | F | O | E | T | | W | T | | T | C | E | S | T | Y | | | |
| | | | | | | | S | O | I | P | T | | | C | R | | | | P | T | | | | | |
| | | | | | | | D | G | | | | | | | | | | | | | | | | | |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | BE769E51 | EE631E87 | F | 0 | 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 3EBC8D75 | 30548DD5 | F | 0 | 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | BEFEE4FD | 234A4926 | F | 0 | 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 3F3BDB69 | 6E8987AC | F | 0 | 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | BF741839 | EB88156E | F | 0 | 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 3FA5BE29 | 8ADF0369 | F | 0 | 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | BFCF5E46 | 537AB906 | F | 0 | 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 1 | 3 | 9F | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 3FE6A09E | 667F3BCC | F | 0 | 1 | — | 2 | 0 | 3 | FB | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |

Microcode Table for the Exp(x) Calculation (Concluded)

| P | D | D | P | E | E | C | P | C | C | S | R | H | E | F | I | R | F | S | B | S | T | S | O | O | O |
|---|----------|----------|---|---|---|---|---|---|---|----|---|---|---|---|-----|---|---|---|---|---|---|---|---|---|---|
| A | A | B | B | N | N | L | I | L | O | E | E | A | N | L | N | N | A | R | Y | E | E | E | E | E | E |
| | | | | A | B | K | P | K | N | L | S | L | C | O | S | D | S | C | T | L | S | L | Y | S | C |
| | | | | | | C | E | M | F | O | E | T | | W | T | | T | C | E | S | T | | | | |
| | | | | | | | S | O | I | P | | | | C | R | | | P | T | | | | | | |
| | | | | | | | D | G | | | | | | | | | | | | | | | | | |
| F | 00000409 | 00000014 | F | 1 | 1 | | 2 | 0 | 1 | FF | 1 | 1 | 1 | 0 | 228 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 1 | | 2 | 0 | 3 | FF | 1 | 1 | 0 | 0 | 300 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 40900000 | 00000000 | F | 0 | 0 | | 2 | 0 | 2 | DF | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 1 | 0 | | 2 | 0 | 2 | DF | 1 | 1 | 1 | 0 | 1C0 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | | 2 | 0 | 3 | FF | 1 | 1 | 1 | 0 | 180 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | | 2 | 0 | 3 | FF | 1 | 1 | 1 | 0 | 300 | 0 | 0 | 0 | 0 | 3 | 3 | 1 | 0 | 0 | 0 |
| F | 00000000 | 00000000 | F | 0 | 0 | | 2 | 0 | 3 | FF | 1 | 1 | 1 | 0 | 300 | 0 | 0 | 0 | 0 | 3 | 3 | 0 | 0 | 0 | 0 |

High-Speed Vector Math and 3-D Graphics

Introduction

The Texas Instruments SN74ACT8847 Floating-Point Processor is designed to execute high-speed, high-accuracy mathematical computations. The device is especially suited for matrix manipulations such as those used in graphics or digital signal processing. This FPU multiplies and adds data elements by executing sequences of microprogrammed calculations to form new matrices. The device may be configured for either single- or double-precision operation. Single-precision operation is assumed throughout this report.

The 'ACT8847 can perform integer and logical operations and has built-in, hardwired algorithms for division and square root operations.

This application report outlines the timing, data flow, and programming for several common data vector calculations and matrix transformations. Further, it illustrates some of the programming "tricks" resulting in fastest operation. Throughout, this document compares the timing schemes for programs in which all registers, including the ALU and multiplier internal pipeline registers, are enabled ("pipelined" mode) with those for equivalent programs in which the internal pipeline registers are disabled ("unpiped" mode). Equations are provided to help the programmer select the more efficient mode, and performance figures are included with times given for 30-MHz operations.

This report begins by covering simple vector arithmetic operations, which are categorized as "computational" or "compare" functions for convenience. This document then compares these operations as they are used in graphics applications to perform three-dimensional coordinate transformations, perspective viewing, and clipping.

SN74ACT8847 Floating-Point Units

The 'ACT8847 floating-point unit (FPU) combines a multiplier and an arithmetic-logic unit (ALU) in a single microprogrammable VLSI device. This device is implemented in TI's advanced 0.8- μ m EPIC™ CMOS technology and is fully compatible with the IEEE standard for binary floating-point arithmetic, STD 754-1985, for either single- or double-precision operation.

Instruction inputs can select independent ALU operation, independent multiplier operation, or simultaneous ALU/multiplier operation. Each FPU can handle three types of data input formats. The ALU accepts data operands in integer format or IEEE floating-point format.

EPIC is a trademark of Texas Instruments Incorporated.

Data enters the 'ACT8847 through two 32-bit data buses, DA and DB (see Figure 3), which can be configured to operate as a single 64-bit data bus for double-precision operations. Data can be latched in a 64-bit temporary register or loaded directly into the input registers, RA and RB, which pass data to the multiplier and ALU.

A clock-mode control allows the temporary register to be clocked on the rising or falling edge of the clock to support double-precision ALU operations at the same rate as single-precision operations. Using the temporary register, double-precision numbers on a single 32-bit input bus can be loaded in one clock cycle.

The input registers RA and RB are the first of three levels of internal data registers. Additionally, the ALU and multiplier each have an internal pipeline register and an output register. The ALU's output register is denoted by "S" (sum), and the multiplier's output register is denoted by "P" (product). Any or all of these internal registers may be bypassed.

A 64-bit constant register (C) with a separate clock is provided for temporary storage of a multiplier result, ALU result, or constant for feedback to the multiplier and ALU. An instruction register and a status register are also included.

Four multiplexers select the multiplier and ALU operands from the input, C, S, or P registers. Results are output on the 32-bit Y bus; a Y output multiplexer selects the most or least significant half of the result for output.

The 'ACT8847 FPU is fully compatible with IEEE Standard 754-1985 for addition, subtraction, multiplication, division, square root, and comparison. The 'ACT8847 FPU also performs integer arithmetic, logical operations, and logical shifts. Additionally, absolute value conversions and floating point-to-integer and integer-to-floating-point conversions are available.

For the 'ACT8847, the ALU and multiplier can operate in parallel to perform sums of products and products of sums. Detailed information regarding the instruction inputs for the various 'ACT8847 configurations and operations is given in this section.

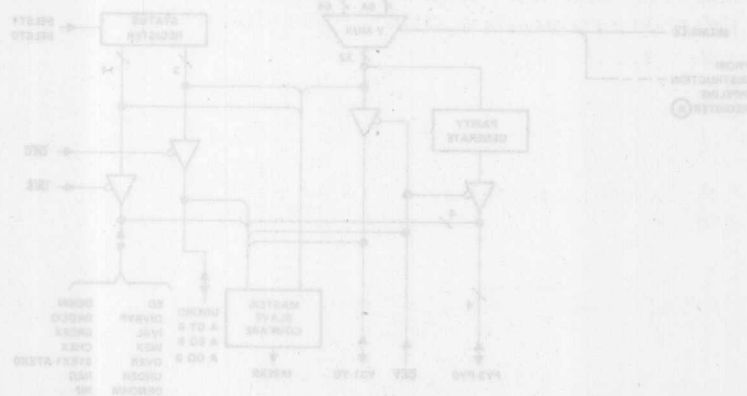


Figure 3. ACT8847 Floating-Point Unit

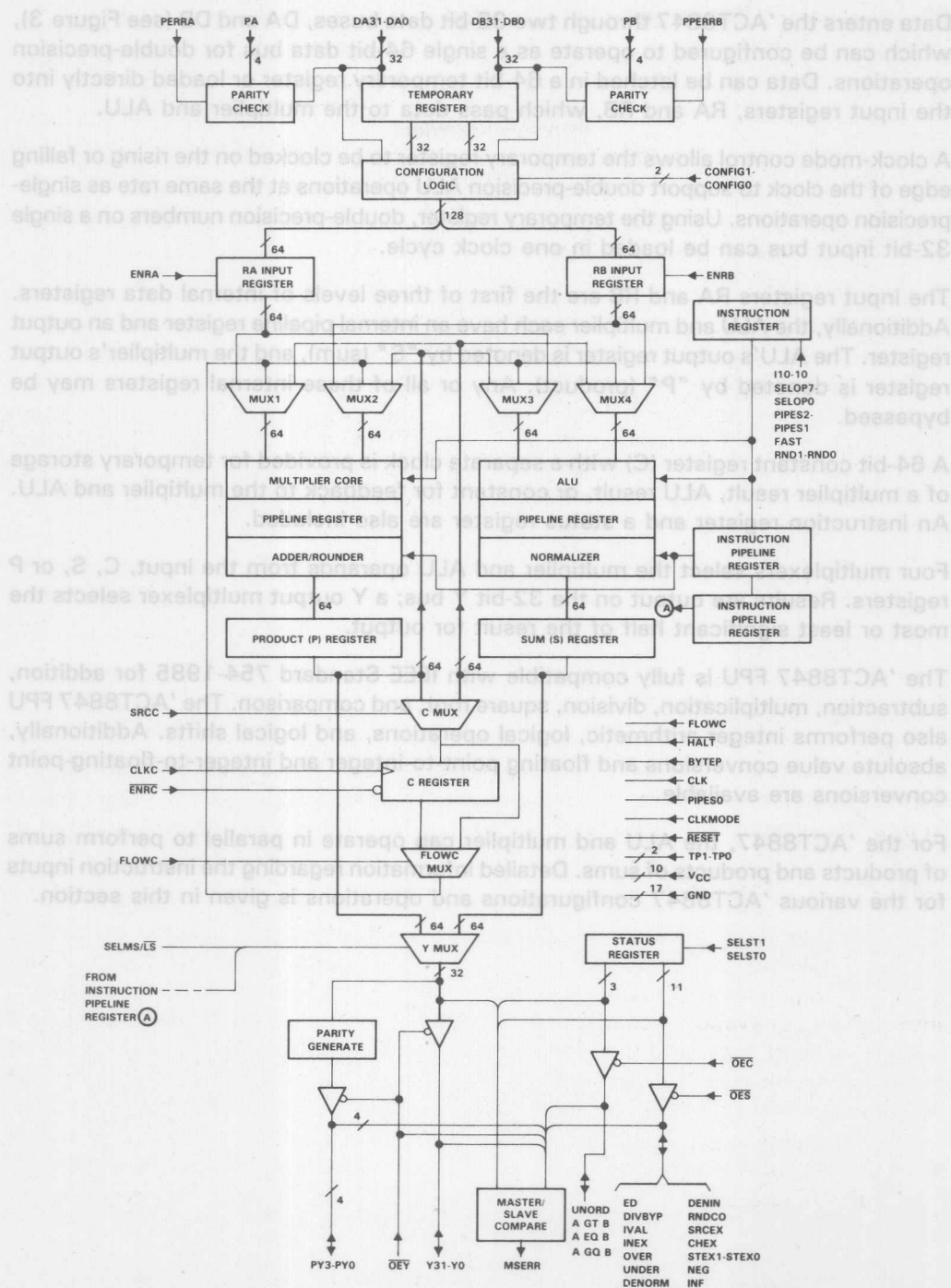


Figure 3. SNACT8847 Floating-Point Unit

Mathematical Processing Applications

TI's SN74ACT8847 high-speed floating-point unit (FPU) is designed to perform high-accuracy, computationally-intensive mathematical operations. In particular, this FPU can meet the computational demands of high-end graphics workstations and advanced signal processing. Both applications involve repetitive computations on arrays of data typically expressed as vector arithmetic operations.

For example, the calculation of the sum of products, or multiply-accumulate function, is frequently used in both signal and graphics processing. In general form, the sum of products equation is:

$$S = \sum_{i=1}^n k_i x_i, \text{ for coefficients } k_i \text{ and data } x_i.$$

This sum of products is the central function involved in multiplying matrices. Such matrices might represent a system of linear differential equations or the geometrical transformation of a graphic object. Specifically, an $n \times n$ matrix A multiplied by an $n \times m$ matrix B yields an $n \times m$ matrix C whose elements c_{ij} are given by:

$$c_{ij} = \sum_{k=1}^n a_{ik} \times b_{kj} \text{ for } i = 1, \dots, n \text{ and } j = 1, \dots, m.$$

The 'ACT8847 is designed to handle efficiently this kind of parallel multiplication and addition.

Graphics Applications

The basic principle of graphics processing is that any object can be reduced to a combination of points, lines, and polygons and then defined as a collection of points in three-dimensional space. Because points, planes, transformation matrices and other common data structures are vectors, most of the computations involved in graphics processing are vector operations.

Computations for a 3-D graphics display are highly involved due to the complexity introduced by the z-axis. Viewing an object from a particular perspective involves transforming the object's world coordinates, or its coordinates in the model space, into viewing, or eyepoint, coordinates. A series of translations and rotations map the viewing system axes onto the world coordinate axes. Each individual point must be translated, rotated and, if necessary, scaled in a proper order. Once the coordinate transformation is complete, the coordinates are clipped to a viewing volume. Clipping algorithms employ arithmetic operations to determine whether an object, or part of an object, is inside or outside a pyramidal volume. Hidden surface routines may then be employed to delete surfaces that fall behind a "nearer" surface from the viewer's perspective.

Matrix arithmetic is required for scaling, rotating, translating, or shearing an object, as well as for the final process of projecting its visible parts to a two-dimensional frame buffer. Any sequence of these transformations can be represented as a single matrix formed by concatenating the matrices for the individual operations. The generalized 4×4 matrix for transforming a three-dimensional object is shown below, partitioned into four component matrices, each of which produces a specific effect on the image. The 3×3 matrix produces linear transformation in the form of scaling, shearing, and rotation. The 1×3 row matrix produces translation, while the 3×1 column matrix produces perspective transformation with multiple vanishing points. The final single-element 1×1 matrix produces overall scaling.

$$T = \left[\begin{array}{c|c} 3 \times 3 & 3 \times 1 \\ \hline 1 \times 3 & 1 \times 1 \end{array} \right]$$

Overall operation of the matrix T on the position vectors of a graphics object produces a combination of shearing, rotation, reflection, translation, perspective, and overall scaling.

Vector Arithmetic

Programs that require repetitive computations on multiple sets of operands lend themselves to vector-processing algorithms, in which the operands are viewed as succeeding elements of long "data vectors." The next two sections outline the programming for commonly-used vector operations. Most of these examples conclude with a comparison of program timing for pipelined (internal pipeline registers enabled) and unpiped (internal pipeline registers disabled) operation. For convenience, the operations are labeled "computational," which includes simple and compounded adds, multiplies, and divides, or "compare," which can be used to select maximum or minimum values from succeeding pairs of numbers or from a list.

Computational Operations on Data Vectors

This section covers the following vector operations: vector add, vector multiply, vector divide, sum of products (also called inner, scalar, or dot product), and product of sums. Since matrix multiplication is composed of a sequence of sum of products operations, these two functions are discussed in the same section. In some cases, a whole class of operations is covered under one heading. For example, the vector add operation includes sums and differences of A_i , B_i , $|A_i|$, and $|B_i|$ in all combinations.

Vector Add

The vector add operation adds corresponding components of data vectors to obtain the components of the output vector. Hence, for input vectors A and B and output vector Y, each with N components,

$$Y_i = A_i + B_i, \quad 1 \leq i \leq N.$$

The 'ACT8847 performs this calculation in unchained, independent ALU mode.

Table 13 shows the contents of the data registers at successive clock cycles for $N = 6$ with the FPU operating in pipelined mode. Since the data travels by way of the internal pipeline register, two cycles pass before the first sum appears in the S register. The contents of the internal pipeline register are not given in the flow.

Table 13. Data Flow for Pipelined Single-Precision Vector Add, $N = 6$

| | | | | | | | | | |
|-----|----|----|---------|---------|---------|---------|---------|---------|---|
| RA | A1 | A2 | A3 | A4 | A5 | A6 | | | |
| RB | B1 | B2 | B3 | B4 | B5 | B6 | | | |
| S | | | A1 + B1 | A2 + B2 | A3 + B3 | A4 + B4 | A5 + B5 | A6 + B6 | |
| P | | | | | | | | | |
| C | | | | | | | | | |
| Y | | | Y1 | Y2 | Y3 | Y4 | Y5 | Y6 | |
| CLK | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Data transfers and operations for each clock cycle are summarized in the program listing in Table 14. Detailed information on the instruction inputs required to perform each operation is included in sections 5 and 7. Note that the selection of the output source (in this case, the S register), which is determined by the I6 instruction bit, is programmed along with the ALU or multiplier operation that generates the output.

Table 14. Program Listing for Pipelined Single-Precision Vector Add, $N = 6$

| REGISTER TRANSFERS | ALU OPERATION | MULTIPLIER OPERATION |
|----------------------------------|---------------|----------------------|
| 1. LOAD RA, RB; $Y \leftarrow S$ | ADD(RA, RB) | |
| 2. LOAD RA, RB; $Y \leftarrow S$ | ADD(RA, RB) | |
| 3. LOAD RA, RB; $Y \leftarrow S$ | ADD(RA, RB) | |
| 6. LOAD RA, RB; $Y \leftarrow S$ | ADD(RA, RB) | |

Timing and programming are similar for other independent ALU operations involving two operands, such as $(A - B)$, $(B - A)$, and compare (A, B) . However, when the compare function is used, two status bits must be generated before numeric values can be output (see "Compare Operations on Data Vectors").

Because the vector add program closely parallels that for vector multiplication, pipelined and unpiped modes for both vector add and multiply are compared in the next section.

Vector Multiply

The vector multiply operation multiplies corresponding elements of data vectors to obtain the components of the output vector. Hence, for input vectors A and B and output vector Y, each with N components,

$$Y_i = A_i \times B_i, \quad 1 \leq i \leq N.$$

The 'ACT8847 performs this calculation in unchained, independent multiplier mode.

Pipelined Mode

Table 15 shows the contents of the data registers at successive clock cycles for N = 6 with the FPU operating in pipelined mode. The product may be replaced by a variety of other independent multiplier operations, such as $-(A \times B)$, $A \times |B|$, $-(A \times |B|)$, $|A| \times |B|$, and $-(|A| \times |B|)$. Data transfers and operations for each clock cycle are summarized in the program listing in Table 16.

Table 15. Data Flow for Pipelined Single-Precision Vector Multiply, N = 6

| | | | | | | | | | |
|-----|----|----|-------|-------|-------|-------|-------|-------|---|
| RA | A1 | A2 | A3 | A4 | A5 | A6 | | | |
| RB | B1 | B2 | B3 | B4 | B5 | B6 | | | |
| S | | | | | | | | | |
| P | | | A1×B1 | A2×B2 | A3×B3 | A4×B4 | A5×B5 | A6×B6 | |
| C | | | | | | | | | |
| Y | | | Y1 | Y2 | Y3 | Y4 | Y5 | Y6 | |
| CLK | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Table 16. Program Listing for Pipelined Single-Precision Vector Multiply, N = 6

| REGISTER TRANSFERS | ALU OPERATION | MULTIPLIER OPERATION |
|---------------------|---------------|----------------------|
| 1. LOAD RA, RB; Y←P | | MULT(RA,RB) |
| 2. LOAD RA, RB; Y←P | | MULT(RA,RB) |
| 3. LOAD RA, RB; Y←P | | MULT(RA,RB) |
| . | | |
| 6. LOAD RA, RB; Y←P | | MULT(RA,RB) |

Unpipied Mode

Table 17 shows the contents of the data registers at successive clock cycles during a vector multiply operation for $N = 6$ with the FPU operating in unpiped mode. The vector add operation progresses similarly. Since there is no "single-clocked storage" in the internal pipeline register, each product or sum is performed in one cycle.

Table 17. Data Flow for Unpipied Single-Precision Vector Multiply, $N = 6$

| | | | | | | | | | |
|-----|----|----------------|----------------|----------------|----------------|----------------|----------------|---|---|
| RA | A1 | A2 | A3 | A4 | A5 | A6 | | | |
| RB | B1 | B2 | B3 | B4 | B5 | B6 | | | |
| S | | | | | | | | | |
| P | | $A1 \times B1$ | $A2 \times B2$ | $A3 \times B3$ | $A4 \times B4$ | $A5 \times B5$ | $A6 \times B6$ | | |
| C | | | | | | | | | |
| Y | | Y1 | Y2 | Y3 | Y4 | Y5 | Y6 | | |
| CLK | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Comparison of Pipelined and Unpipied Modes

For both vector add and vector multiply operations carried out in pipelined mode, results are output to the Y bus on clocks 3,..., $N + 2$. In unpiped mode, results are output to the Y bus on clocks 2,..., $N + 1$, thereby saving a cycle. Unfortunately, it is necessary to operate at a lower clock rate in unpiped mode than in pipelined mode. The following equation can be used to determine which of the two modes provides the faster performance in a particular application. Pipelined operation is faster if:

$$(N + 2)/F_p < (N + 1)/F_u,$$

where F_p and F_u are the clock rates in pipelined and unpiped modes, respectively. As of publication, pipelined mode provides faster performance for input vectors with $N > 2$.

Sum of Products

The sum of products operation multiplies corresponding elements of data vectors and adds the resulting products. The operation is also referred to as the inner product, scalar product, or dot product of two vectors, since these are the names for the function as it is used in vector algebra. For input vectors A and B, each with N components, the sum of products operation yields a single output Y defined as follows:

$$Y = \sum_{i=1}^N A_i \times B_i$$

The 'ACT8847 performs this calculation in chained mode so that concurrent operation of the ALU and multiplier is possible.

Pipelined Mode

Table 18 shows the contents of the data registers at successive clock cycles for $N = 8$ with the FPU operating in pipelined mode.

Table 18. Data Flow for Pipelined Single-Precision Sum of Products, $N = 8$

| | | | | | | | | | | | | | | | |
|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|
| RA | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | | | | | | | |
| RB | B1 | B2 | B3 | B4 | B5 | B6 | B7 | B8 | | | | | | | |
| S | | | | | S1 | | S3 | S4 | S5 | S6 | S7 | S8 | | | S7 + 8 |
| P | | | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | | | | | |
| C | | | | | P2 | P2 | | | | | | | | | |
| Y | | | | | | | | | | | | | | | Y |
| CLK | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | |

Here, $P_i = A_i \times B_i$, $S_1 = P_1 + 0$, $S_3 = P_3 + S_1$, $S_4 = P_4 + P_2$, $S_6 = P_6 + S_4$, $S_7 = P_7 + S_5$, and $S_8 = P_8 + S_6$. The values of the sums could be more succinctly expressed as $S_i = P_i + S_{i-2}$ (with $S_0 = S_{-1} = 0$), except that $S_2 = P_2 + 0 = P_2$ does not actually appear in the data flow as a sum in the S register. Instead, the C register holds P_2 for two cycles.

This approach, although introducing a certain lack of symmetry into the programming, frees up the S register at a point allowing the efficient overlap of succeeding sum of products operations without any dead cycles. A new sum of products operation can begin at CLK 9, and the S register remains free to hold the first operation's result in CLK 14. Similarly, by storing S_7 in the C register in CLK 12, rather than multiplying it by one, the P register remains free to hold "P2" for the next pair of data vectors. By CLK 12, $S_7 = P_1 + P_3 + P_5 + P_7$ and $S_8 = P_2 + P_4 + P_6 + P_8$, so that $Y = S_7 + S_8$.

Data transfers and operations for each clock cycle are summarized in the program listing in Table 19.

Table 19. Program Listing for Pipelined Single-Precision Sum of Products, $N = 8$

| REGISTER TRANSFERS | ALU OPERATION | MULTIPLIER OPERATION |
|---------------------|---------------|----------------------|
| 1. LOAD RA, RB | | MULT(RA,RB) |
| 2. LOAD RA, RB | | MULT(RA,RB) |
| 3. LOAD RA, RB | ADD(P,0) | MULT(RA,RB) |
| 4. LOAD RA, RB; C←P | | MULT(RA,RB) |
| 5. LOAD RA, RB | ADD(P,S) | MULT(RA,RB) |
| 6. LOAD RA, RB | ADD(P,C) | MULT(RA,RB) |
| 7. LOAD RA, RB | ADD(P,S) | MULT(RA,RB) |
| 8. LOAD RA, RB | ADD(P,S) | MULT(RA,RB) |
| 9. | ADD(P,S) | |
| 10. | ADD(P,S) | |
| 11. C←S | | |
| 12. Y←S | ADD(S,C) | |

The above algorithm imposes no delay between input vectors. The time required to carry out the sum of products operation on M pairs of input vectors in succession, each of length N , is $N \times M + 6$ cycles.

Unpiped Mode

In the unpiped version of the sum of products, the data flow is more straightforward. Again, chained mode is employed to allow the ALU and multiplier to operate concurrently. Table 20 shows the contents of the data registers at successive clock cycles for $N = 8$ with the FPU operating in unpiped mode. Here, $P_i = A_i \times B_i$, and $S_i = S_{(i-1)} + P_i$, with $S_0 = 0$.

Table 20. Data Flow for Unpiped Single-Precision Sum of Products, $N = 8$

| | | | | | | | | | | | | | | |
|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RA | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | | | | | | |
| RB | B1 | B2 | B3 | B4 | B5 | B6 | B7 | B8 | | | | | | |
| S | | | S1 | S2 | S3 | S4 | S5 | S6 | S7 | S8 | | | | |
| P | | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | | | | | |
| C | | | | | | | | | | | | | | |
| Y | | | | | | | | | | Y | | | | |
| CLK | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

A new problem can be presented at CLK 9 without any delay between the vectors. Therefore, the time required to compute the sums of products for M pairs of vectors, each of length N , is $N \times M + 2$ clock cycles.

Comparison of Pipelined and Unpipelined Modes

The following equation can be used to determine which of the two modes provides the faster performance in a particular application. Pipelined operation is faster if:

$$(M \times N + 6)/F_P < (M \times N + 2)/F_U,$$

where F_P and F_U are the clock rates in pipelined and unpipelined modes, respectively. Because the unpipelined mode's longer clock cycle usually outweighs its savings in cycles, pipelined mode provides faster performance for input vectors with $N > 4$.

Product of Sums

The product of sums operation adds corresponding elements of data vectors and multiplies the resulting sums. For input vectors A and B, each with N components, the product of sums operation yields a single output Y defined as follows:

$$Y = \prod_{i=1}^N (A_i + B_i)$$

The product of differences can be computed by simply making the ALU operation $(A - B)$ or $(B - A)$. The 'ACT8847 performs this calculation in chained mode so that concurrent operation of the ALU and multiplier is possible. The data flow and program listing for the product of sums are identical to those for the sum of products, except that the roles of add and multiply are reversed. The criteria used to decide between pipelined and unpipelined modes are also identical to those previously given.

Vector Divide

The vector divide operation divides corresponding elements of data vectors to obtain the components of the output vector. Hence, for vectors A and B and output vector Y, each with N components,

$$Y_i = A_i/B_i, \quad 1 \leq i \leq N.$$

The 'ACT8847 performs this calculation using the Newton-Raphson iterative method. In the 'ACT8847, the divide algorithm is built in.

A General Principle

The vector divide example illustrates a general programming principle that should be considered whenever a program begins a new instruction every other cycle. In cases where the C register is not used, it is simple to interleave another program, even one not performing the same function.

Interleaving programs is not as easy if the C register is used because the C register is the only nonpipelined register. However, even using the C register, programs may often be interleaved by staggering one against the other so that their use of the C register

does not overlap in time. Many of the programs so far discussed can be thought of as two such interleaved programs, with the C register being used to delay the first result until it can be combined with the second. (See, for example, the sum of products operation.)

SN74ACT8847 Vector Divide

Since the 'ACT8847 has a built-in algorithm for divide, Table 21 shows the data flow for pipelined operation. Data transfers and operations are summarized in the program listing in Table 22.

Table 21. Data Flow for 'ACT8847 Pipelined Single-Precision Vector Divide

| | | | | | | | | | | | |
|-----|----|---|---|---|---|---|-------|---|---|----|--|
| RA | A1 | | | | | | A2 | | | | |
| RB | B1 | | | | | | B2 | | | | |
| S | | | | | | | | | | | |
| P | | | | | | | A1/B1 | | | | |
| C | | | | | | | | | | | |
| Y | | | | | | | Y1 | | | | |
| CLK | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |

Table 22. Program Listing for 'ACT8847 Pipelined Single-Precision Vector Divide

| REGISTER TRANSFERS | | | | ALU OPERATION | | | | MULTIPLIER OPERATION | | | |
|--------------------|--------------|-----|--|---------------|--|--|--|----------------------|--|--|--|
| 1. | LOAD RA, RB; | Y←P | | | | | | DIVIDE | | | |
| 7. | LOAD RA, RB; | Y←P | | | | | | DIVIDE | | | |
| 13. | LOAD RA, RB; | Y←P | | | | | | DIVIDE | | | |

Note that the microinstructions are presented on the steps indicated (1, 7, 13,...), with a six-cycle lapse before the next operands can be input to RA and RB. Performing a vector divide of two N-element single-precision vectors takes (6N + 2) cycles in pipelined mode. M such pairs of vectors would require [6(N × M) + 2] cycles in pipelined mode. In unpiped mode, the equation is 7(N × M).

Compare Operations on Data Vectors

In independent ALU mode (unchained), two operands may be compared for equality ($A = B$) and order ($A > B$). Additionally, the absolute values of either or both operands may be compared. The compare function uses two status bits, the AGTB and AEQB output signals. (When any operation other than a compare is performed, either by the ALU or the multiplier, the AEQB signal is used as a zero detect. Hence, numerical results cannot be output in the same cycle in which comparison status is output.)

For greatest efficiency, programs for compare operations should be written without requiring conditional branches in the sequencer. If branches can be avoided, the microcoding is simplified and the programs are immediately scalable to SIMD systems employing many 'ACT8847 chips.

This section covers vector max/min and list max/min operations.

Vector MAX/MIN

The vector max/min operations compare corresponding elements of data vectors and select the maximum or minimum value to obtain the components of the output vector. Hence, for input vectors A and B and output vector Y, each with N components,

$$Y_i = \text{MAX/MIN}(A_i, B_i), \quad 1 \leq i \leq N.$$

Pipelined Mode

Table 23 shows the suggested data flow for a pipelined vector MAX operation, where Y_i is set to the max of (A_i, B_i) for all i . Included are rows to indicate the setting of the chain mode instruction bit (I10 for the 'ACT8847) and the status bit being sensed.

Table 23. Data Flow for Pipelined Single-Precision Vector MAX

| CHAIN | N | Y | Y | Y | N | Y | Y | Y | N | Y |
|--------|----|----|-----|----|----|----|-----|----|----|----|
| RA | A | A1 | | B1 | A2 | A2 | | B2 | A3 | A3 |
| RB | B1 | | | | B2 | | | | B3 | |
| S | | | | A1 | | B1 | | A2 | | B2 |
| P | | | | | | A1 | | | | A2 |
| C | | | | | | | | | | |
| Y | | | | | | Y1 | | | | Y2 |
| STATUS | | | A>B | | | | A>B | | | |
| CLK | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

A comparison starts at $\text{CLK} = 1, 5$, etc., when the chain-mode instruction bit is low. The result appears at $\text{CLK} = 3, 7$, etc., indicated by the AGTB and AEQB signals. AGTB is saved off-chip for use as instruction bit I6 (output source) at $\text{CLK} 4, 8$, etc. This value for I6 selects the output source, either the multiplier or the ALU result, at $\text{CLK} 6, 10$, etc. For example, if a comparison result is $A > B$, the AGTB signal goes

high and is used to set I6 high. I6 then selects the multiplier result (A_i) to output. Similarly, if $A \leq B$, AGTB and I6 are low, and the ALU result (B_i) is output. The circuitous route taken by A_i on the way to the P register is necessary because it is not possible to pass RA or RB through the multiplier in parallel with passing the other through the ALU.

The program is not particularly well-packed and produces the vector max of a pair of vectors of length N in $(4N + 2)$ cycles. For M pairs of vectors of length N, the total time is $(4MN + 2)$ cycles. The program can be improved by applying the interleaving principle previously discussed. The steps are rearranged so that a new operation begins every other cycle, thus allowing two compare programs to be interleaved. Table 24 shows the suggested data flow for a pipelined vector min/max operation, where $Y_i = \text{MAX/MIN}(A_i, B_i)$ and $Z_i = \text{MAX/MIN}(C_i, D_i)$.

Table 24. Data Flow for Pipelined Single-Precision Interleaved Vector MAX/MIN

| CHAIN | N | N | Y | Y | Y | Y | N | N | Y | Y | Y | Y | N | N |
|--------|----|----|-------|-------|----|----|----|----|-------|-------|----|----|----|----|
| RA | A1 | C1 | A1 | C1 | B1 | D1 | A2 | C2 | A2 | C2 | B2 | D2 | | |
| RB | B1 | D1 | | | | | B2 | D2 | | | | | | |
| S | | | | | A1 | C1 | B1 | D1 | | | A2 | C2 | B2 | D2 |
| P | | | | | | | A1 | C1 | | | | | A2 | C2 |
| C | | | | | | | | | | | | | | |
| Y | | | | | | | Y1 | Z1 | | | | | Y2 | Z2 |
| STATUS | | | A > B | A > B | | | | | A > B | A > B | | | | |
| CLK | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

Again, A_i (and C_i) reaches the P register by an indirect route. However, this tighter program performs M vector comparisons, two vector comparisons at a time, in $[6 \times N \times \text{CEILING}(M/2) + 2]$ cycles. (As previously defined, the ceiling function rounds to the next highest integer for fractional values.) In this example, two separate vector comparisons on two-dimensional vectors are performed, giving $6 \times 2 \times 1 + 2 = 14$ cycles. For $M = 2$ pairs of vectors, all of length N, the second program is as good as the first. For $M > 2$, the interleaved program performs increasingly better as M gets larger.

This second program requires more off-chip logic, since the status outputs at CLK 3 and 4 must be saved separately off-chip for use at CLK 5 and 6, respectively. This problem can easily be avoided by starting the calculations on the second pair of vectors two cycles later than shown (i.e., at CLK 4). The time necessary to perform the vector MAX operation on M pairs of N-dimensional vectors, two pairs concurrently, then increases to $[6 \times N \times \text{CEILING}(M/2) + 4]$ cycles.

Data transfers and operations for the odd lines only are summarized in the program listing in Table 25. The complete program is obtained by repeating the equivalent of each odd-numbered line in the next even line for the second pair of vectors.

Table 25. Program Listing for Pipelined Single-Precision Interleaved Vector MAX/MIN

| REGISTER TRANSFERS | ALU OPERATION | MULTIPLIER OPERATION |
|--------------------|----------------|----------------------|
| 1. LOAD RA, RB | COMPARE(RA,RB) | |
| 3. LOAD RA | ADD(RA,0) | |
| 5. LOAD RA; Y←P/S | ADD(RA,0) | MULT(S,1) |

Unpiped Mode

Table 26 shows the data flow for an unpiped vector MAX operation.

Table 26. Data Flow for Unpiped Single-Precision Vector MAX

| CHAIN | N | Y | Y | N | Y | Y | N | Y | Y |
|--------|----|-----|----|----|-----|----|----|-----|----|
| RA | A1 | A1 | B1 | A2 | A2 | B2 | A3 | A3 | B3 |
| RB | B1 | | | B2 | | | B3 | | |
| S | | | A1 | B1 | | A2 | B2 | | A3 |
| P | | | | A1 | | | A2 | | |
| C | | | | | | | | | |
| Y | | | | Y1 | | | y2 | | |
| STATUS | | A>B | | | A>B | | | A>B | |
| CLK | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

The status bit is saved off-chip at CLK = 2, 5, etc., and used at CLK = 3, 6, etc., as the I6 bit of the instruction. I6 selects either the multiplier or ALU result to output to the Y bus at CLK = 4, 7, etc.

The program computes the vector comparison of M pairs of vectors of length N in $[3 \times M \times (N + 1)]$ cycles.

Comparison of Pipelined and Unpiped Operation

Pipelined operation is faster if:

$$[6 \times N \times \text{CEILING}(M/2) + 2]/F_p < (3 \times M \times N + 1)/F_u,$$

where F_p and F_u are the clock rates in pipelined and unpiped modes, respectively. As of publication, pipelined mode provides faster performance for $M > 1$.

List MAX/MIN

The list max/min operations select the maximum or minimum value, Z , of a list of N elements. Hence, for input vector A with N components and output Z ,

$$Z = \text{MAX/MIN}(A_i), \quad 1 \leq i \leq N.$$

List min/max is an essential operation in computer graphics because it is used to find the "extents" of a polygon or polyhedron. The extents are the maximum values of X , Y , and Z among the list of vertices for the object in question. Many forms of comparison are possible since the absolute value of either or both ALU operands may be employed. However, the example in this section assumes that the largest element of a list of N elements is desired.

Pipelined Mode

Table 27 shows the data flow for a pipelined list MAX operation, where $M_1 = \text{MAX}(A_1, A_2)$; $M_i = \text{MAX}[M_{i-1}, A_{(i+1)}]$, $2 \leq i \leq N - 2$.

Table 27. Data Flow for Pipelined Single-Precision List MAX

| CHAIN | Y | N | Y | Y | Y | N | Y | Y | Y | N | Y | Y | Y | Y | Y | Y |
|--------|----|----|----|-------|----|----|----|-------|----|----|----|-------|----|----|----|----|
| RA | A1 | A1 | A2 | | | A3 | A3 | | | A4 | A4 | | | | | |
| RB | | A2 | | | | | | | | | | | | | | |
| S | | | A1 | | A2 | | | | M1 | | | | M2 | | | M3 |
| P | | | | | A1 | | | | A3 | | | | A4 | | | |
| C | | | | | | M1 | M1 | | | M2 | M2 | | | M3 | | |
| Y | | | | | | | | | | | | | | | | M3 |
| STATUS | | | | A > B | | | | A > B | | | | A > B | | | | |
| CLK | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |

As with vector comparison, the max/min of the absolute values is available, since the chip operates in independent ALU mode on the comparison steps. The comparison is between the RA register and the RB register in step 2 and between RA and C in steps 6, 10, etc. In these steps, the chip is switched into unchained, independent ALU mode. The status is saved off-chip and used to set the SRCC signal, which selects whether the P or S data goes into the C register in steps 5, 9, etc.

When the list max is in the C register, at $\text{CLK} = 4N - 2$, the C register contents must then be passed through one of the functional units to the output. The MAX/MIN of an N -element list therefore takes $4N$ cycles. M such vectors can be processed in $[M(4N - 1) + 1]$ cycles.

Data transfers and operations for the list max operation are summarized in the program listing in Table 28. The program is carried out in pipelined mode, alternating between unchained and chained modes. The list max reaches the output in cycle $4N$.

Table 28. Program Listing for Pipelined Single-Precision List MAX

| REGISTER TRANSFERS | ALU OPERATION | MULTIPLIER OPERATION |
|--|----------------|----------------------|
| 1. LOAD RA | ADD(RA,0) | |
| 2. LOAD RA, RB | COMPARE(RA,RB) | |
| 3. LOAD RA | ADD(RA,0) | MULT(S,1) |
| 4. | | |
| 5. C←P/S | | |
| 6. LOAD RA | COMPARE(RA,C) | |
| 7. LOAD RA | ADD(C,0) | MULT(RA,1) |
| 8. | | |
| 9. C←P/S | | |
| REPEAT STEPS 6 THROUGH 9 UNTIL STEP $4N - 2$ IS REACHED, THEN: | | |
| $4N - 2$ | Y←S | ADD(C,0) |

Comparison of Pipelined and Unpipied Modes

The equivalent unpiped program takes $[M(3N - 1) + 1]$ cycles. Pipelined mode is fastest if:

$$[M(4N - 1) + 1]/F_p < [M(3N - 1) + 1]/F_u,$$

where F_p and F_u are the clock rates in pipelined and unpiped modes, respectively. As of publication, pipelined mode provides faster performance for all M and N .

Graphics Applications

This section summarizes the concepts related to creating a three-dimensional image and examines a few of the matrix operations used in three-dimensional graphics processing. These operations include coordinate transformations and clipping operations. Additionally, this section illustrates some of the programming techniques used to perform these operations.

Creating a 3-D Image

Conceptually, translating 3-D images to 2-D display screens involves defining a view volume that limits the scope of the vista the viewer can see at one time. For simplicity, a standardized frame of reference, in which the viewer's eye is located at the origin of the coordinate system, is adopted in this example.

As illustrated in Figures 4(a) and 4(b), the arbitrary world coordinates of the objects under scrutiny are transformed into normalized "viewing" or "eye" coordinates that reflect this frame of reference. Once the normalizing transformation is complete, the images within the view volume are projected onto a 2-D view plane, which is assumed to be located, like a projection screen, at a suitable relative distance from the viewer (see Figures 4(c) and 5).

A basic model for creating a 3-D view, illustrated in Figure 6(a), transforms arbitrary world coordinates to normalized viewing coordinates and then "clips" the image to remove lines that do not fall within the normalized view volume. Clipping is followed by projecting the image to the 2-D projection plane (or "window"). The image is then mapped onto a canonical 2-D viewport display and from there onto the physical device.

To incorporate image transformations, another model must be adapted [see Figure 6(b)]. After clipping, instead of projecting to the view plane, a perspective transformation is performed on the clipped viewing coordinates, transforming the view volume into a 3-D viewport, the "screen system" in which image transforms are performed. Then the image is projected to the 2-D viewport display and onto the physical device.

In both models, the clipping operation is performed on coordinates in the viewing system. This approach is referred to as "clipping in the eye system." In practice, clipping is often performed after transformation to the screen system. A trivial accept/reject test is performed on viewing coordinates, the image is transformed to the screen system, and then clipping is performed.

(c) (SEE NOTE C)

(c)

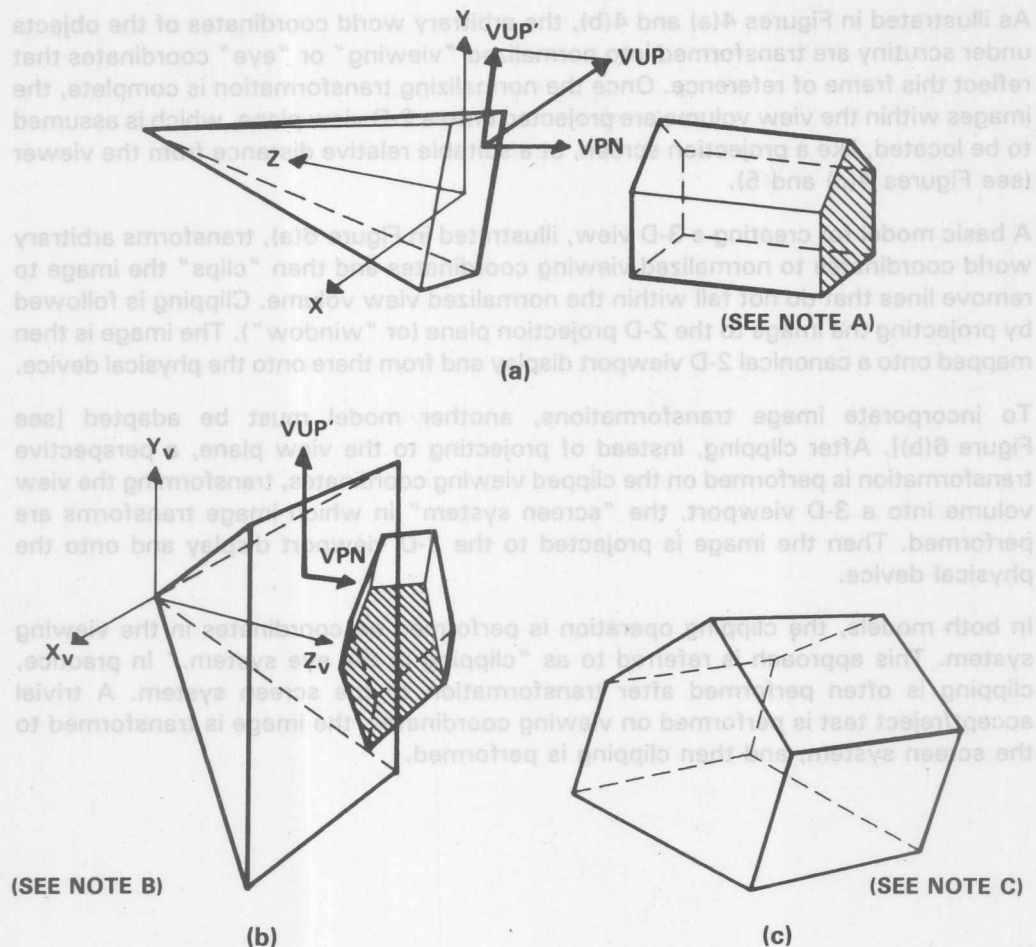
(b) (SEE NOTE B)

(b)

NOTES: A. In a sequence of transformations, the world coordinate positions for the house are transformed into the normalized viewing coordinate system (also called the eye system). For clarity, the house is pictured outside the view column. Also shown are the direction vector VUP (view up), VFN (view normal), and VUP (the projection of VUP parallel to the view plane). B. After a series of translations, rotations, and shearing and scaling operations, the view volume becomes the canonical perspective projection view volume, which is a truncated pyramid with apex at the origin, and the house has been transformed from the world to the viewing coordinate system. C. This figure illustrates the projection of the house from the perspective of the viewer, with eye located at the origin of the coordinate system.

Figure 4. Creating a 3-D image

J. D. Foley and A. Van Dam, Fundamentals of Interactive Computer Graphics, Addison-Wesley Publishing Company, Reading, MA, 1982, 281-283. Reprinted with permission.



- NOTES: A. In a sequence of transformations, the world coordinate positions for the house are transformed into the normalized viewing coordinate system (also called the eye system). For clarity, the house is pictured outside the view column. Also shown are the direction vectors VUP (view up), VPN (view normal), and VUP' (the projection of VUP parallel to VPN onto the view plane).
- B. After a series of translations, rotations, and shearing and scaling operations, the view volume becomes the canonical perspective projection view volume, which is a truncated pyramid with apex at the origin, and the house has been transformed from the world to the viewing coordinate system.
- C. This figure illustrates the projection of the house from the perspective of the viewer, with eye located at the origin of the coordinate system.

Figure 4. Creating a 3-D Image

J. D. Foley and A. Van Dam, Fundamentals of Interactive Computer Graphics , Addison-Wesley Publishing Company, Reading, MA, 1982, 291-293. Reprinted with permission.

The following sections illustrate programming techniques used in both of these approaches to normalizing, clipping, and transforming a 3-D image. The operations are grouped as "3-D Coordinate Transforms," "Clipping in the Eye System," and "Clipping in the Screen System."

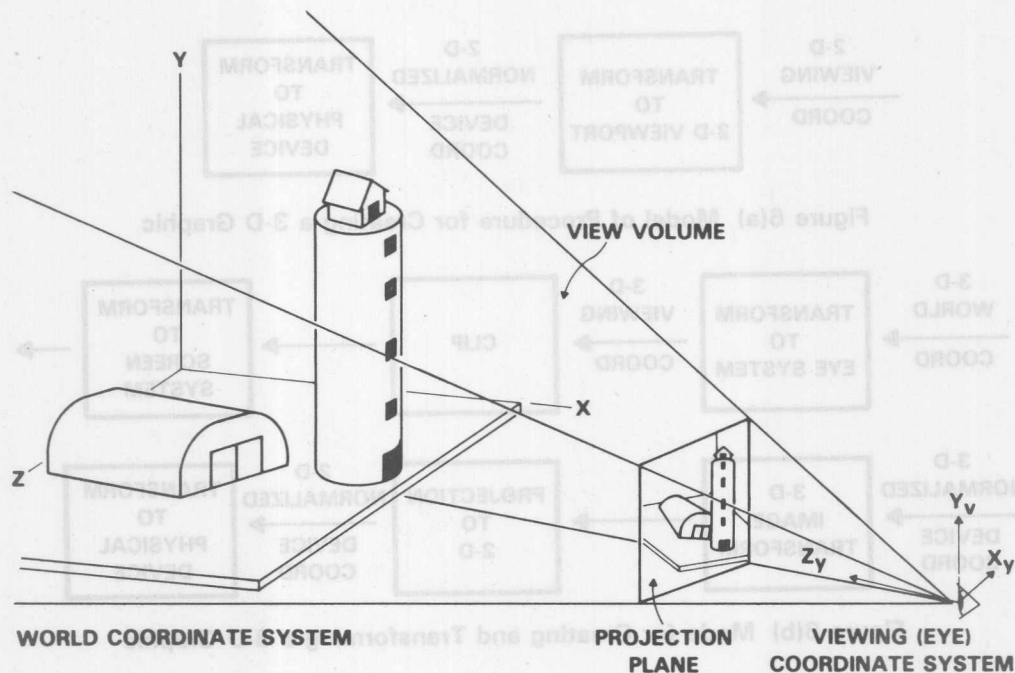


Figure 5. View Volume

Adapted with permission from a paper by Stephen R. Black entitled "Digital Processing of 3-D Data to Generate Interactive Real-Time Dynamic Pictures" from Volume 120 of the 1977 SPIE journal "Three Dimensional Imaging."

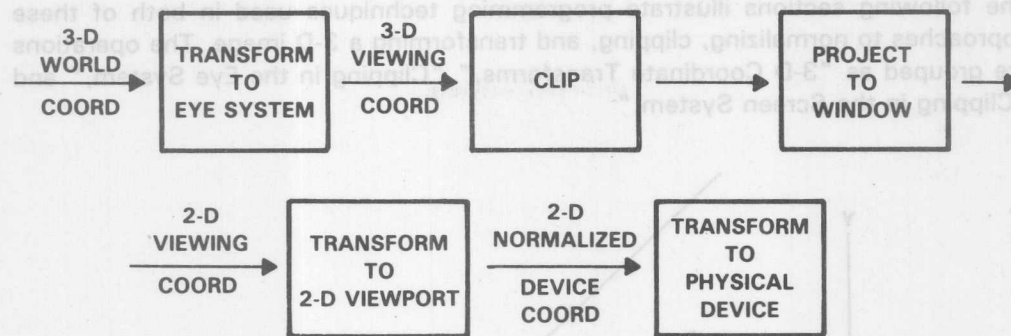


Figure 6(a) Model of Procedure for Creating a 3-D Graphic

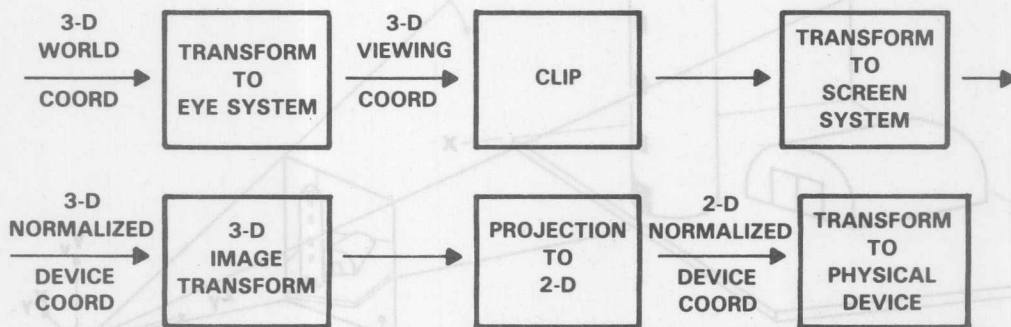


Figure 6(b) Mode for Creating and Transforming a 3-D Graphic

Three-Dimensional Coordinate Transforms

One of the computationally-intensive functions of a 3-D computer graphics system is that of transforming points within the object space, such as translating an object or rotating an object about an arbitrary axis. Equally complex is the transformation of points within the object space (or "world coordinate system") into points defined by a particular perspective and located within the viewing space (or "eye coordinate system"). This latter process, known as the viewing transformation, generates points in a left-handed cartesian system with the eye at the origin and the z-axis pointing in the direction of view. The arbitrary world-system view volume and the objects therein are translated, rotated, sheared, and scaled to match the predefined, canonical view volume of the eye system.

For a "realistic" image, the canonical view volume will be a truncated pyramid that mimics the cone of vision available to the human eye. Alternatively, the volume can be a unit cube. The series of operations that make up each transformation differ, but if homogeneous coordinates are used, either transformation can be expressed as a simple matrix multiply.

For each point (X, Y, Z) in the world system, a projection in homogeneous coordinates is denoted by (X_h, Y_h, Z_h, W_h) where,

$$(X_h, Y_h, Z_h, W_h) = (X \times W_h, Y \times W_h, Z \times W_h, W_h),$$

and W_h is simply a scale factor, typically unity when floating-point numbers are used. (With fixed point values, nonunity values of W_h are used to maximize use of the numeric range.) To transform a point in homogeneous coordinates, it is post-multiplied by a 4 × 4 transform matrix:

$$[X_h', Y_h', Z_h', W_h'] = [X_h, Y_h, Z_h, W_h] \times \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix}$$

The transformed point can later be converted back to 3-space by dividing by W_h:

$$(X', Y', Z') = (X_h'/W_h', Y_h'/W_h', Z_h'/W_h')$$

The transform matrix is constructed by multiplying together a sequence of matrices, each of which performs a simple task. The product of 4 or 5 elementary matrices may be used to perform some complex overall operation on a set of points representing an object or an entire scene. Once constructed, the transform matrix is used on each point of the object to be transformed.

This section describes two approaches to the viewing transformation—the general case and the specific yet typical case in which a reduced version of the transform matrix may be used. Performance times are given for 30-MHz frequencies, which roughly correspond to the operating speeds of the 'ACT8847.

Operation with General Transform Matrix

Table 29 shows part of the data flow for the pipelined and chained program for the product of the homogeneous point [X, Y, Z, W] and the 4 × 4 transform matrix A.

Table 29. Partial Data Flow for Product of [X, Y, Z, W] and General Transform Matrix

| RA | X | Y | Z | W | X | Y | Z | W | X | Y |
|-----|-----|-----|-------|-------|-------|-------|-------|-------|-------|-------|
| RB | A11 | A21 | A31 | A41 | A12 | A22 | A32 | A42 | A13 | A23 |
| S | | | | | S1(1) | | S3(1) | S4(1) | S1(2) | T1 |
| P | | | P1(1) | P2(1) | P3(1) | P4(1) | P1(2) | P2(2) | P3(2) | P4(2) |
| C | | | | | P2(1) | P2(1) | | S3(1) | P2(2) | P2(2) |
| Y | | | | | | | | | | X' |
| CLK | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

The technique is that already illustrated for the sum of products operation. The numbers in parentheses indicate which column of the transform matrix is involved in the operation. Here, $P1(i) = X \times A1i$, $P2(i) = Y \times A2i$, etc. $S1(i) = P1(i) + 0$, $S3(i) = S1(i) + P3(i)$, $S4(i) = P2(i) + P4(i)$, and $Ti = S3(i) + S4(i)$. $T1 = X'$, $T2 = Y'$, $T3 = Z'$, $T4 = W'$. As in the sum of products illustration, in order to make the most efficient use of the S register, P2 is used directly instead of summing by 0 to form S2.

The time to transform N points in a system is $16N + 6$ cycles. The system can transform approximately 1.875 million points per second at a clock rate of 30 MHz.

Operation with the Reduced Transform Matrix and $W_h = 1$

Because viewing transformations are frequently carried out using a single-vanishing-point perspective, the 3×1 column that performs perspective transformations with multiple vanishing points is often not used. Additionally, with $W_h = 1$, the 1×1 scale factor is often equal to one. In these cases, the transform matrix takes the following form:

$$\begin{bmatrix} \dots & 0 \\ \dots & 0 \\ \dots & 0 \\ \dots & 0 \\ \dots & 1 \end{bmatrix}$$

With multiple vanishing points, and in other graphics operations such as clipping, 4×4 matrices are used with nonzero values in the fourth column. The transform matrix is termed "reduced" when its fourth column is the same as that previously shown. In such cases, the transform of each point requires only 9 multiplications and 9 additions.

Table 30 shows part of the data flow for the reduced matrix program.

Table 30. Partial Data Flow for Product of [X, Y, Z, W] and Reduced Transform Matrix

| RA | X | Y | Z | x | X | Y | Z | x | X |
|-----|-----|-----|-------|-------|-------|-------|-------|-------|-------|
| RB | A11 | A21 | A31 | A41 | A12 | A22 | A32 | A42 | A13 |
| S | | | | | | S1(1) | S2(1) | | T1 |
| P | | | P1(1) | P2(1) | P3(1) | | P1(2) | P2(2) | P3(2) |
| | | | | P1(1) | P2(1) | | S1(1) | P1(2) | P2(2) |
| Y | | | | | | | | | X' |
| CLK | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Again, the numbers in parentheses refer to the column of the transform matrix involved in the operation. In this case, however, only the first three columns are used. Hence, for $1 \leq i \leq 3$, $P1(i) = X \times A1i$, $P2(i) = Y \times A2i$, etc. $S1(i) = P1(i) + A4i$, $S2(i) = P2(i) + P3(i)$, and $Ti = S1(i) + S2(i)$. $T1 = X'$, $T2 = Y'$, $T3 = Z'$. Note that W values are not calculated since they are all 1.

The time to transform N points in a system is $(12N + 5)$ cycles. The system can transform 2.5 million points per second at 30 MHz.

Three-Dimensional Clipping

Once an image is transformed into viewing coordinates, it must be clipped so that lines extending outside the view volume are removed. There are several approaches to clipping, some more efficient than others. This section surveys the most commonly used techniques and estimates the throughput of several single- and multi-processor arrangements.

First considered is the technique of fully clipping the line segments to fit within the viewing pyramid in the eye coordinate system. This technique is commonly referred to as "clipping before division."

Clipping in the screen system is considered second. This method eliminates lines that are obviously invisible in the eye system

Clipping in the Eye System

If an object is composed of straight line segments and a perspective view is to be taken, the viewing volume is a pyramid defined by the following plane equations:

$$X = K \times Z, X = -K \times Z, Y = K \times Z, Y = -K \times Z,$$

where K is a constant to be defined below. Thus, $-KZ < (X,Y) < KZ$. Two other clipping planes are usually employed at $Z = N$ and $Z = F$, where N and F are the near and far limits, respectively, of the view. This gives:

$$N < Z < F.$$

Looking in the direction of the z -axis (see Figure 78), the eye can imagine a screen located at a distance N from the eye. K is formed from the half-screen height divided by N . A specific line segment might intersect any or all of the six clipping planes. One common approach to this problem is to use six processors in a pipeline, each clipping the line to one plane.

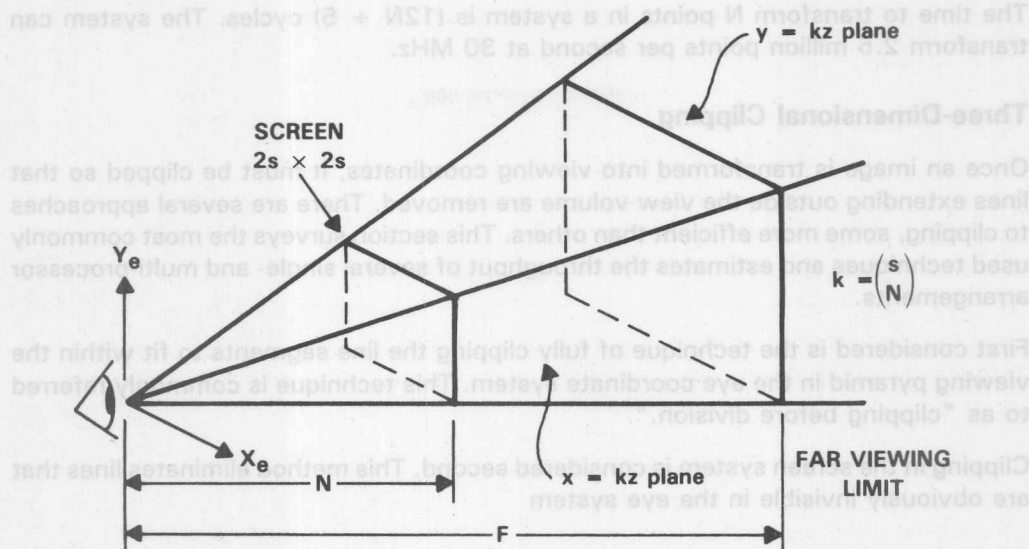


Figure 7. Viewing Pyramid Showing Six Clipping Planes

Consider the case of clipping the line defined by the points $P1 = (X1, Y1, Z1)$ and $P2 = (X2, Y2, Z2)$ against the $Z = N$ plane. First computed are $(Z1 - N)$ and $(Z2 - N)$. If both are negative, the line is invisible, and a notation meaning an empty line is passed on. If both are positive, both ends of the line are on the visible side of the $Z = N$ plane, and the line is passed on unclipped.

When one of these computed values is negative and the other positive, the line must be clipped and the new values for its endpoints passed down the rest of the pipeline. To do so, a parameter t that indicates what fraction of a segment $Z1Z2$, and therefore of $P1P2$ as a whole, lies on the $P1$ side of the $Z = N$ plane, is computed as follows:

$$t = (Z1 - N)/(Z1 - Z2).$$

In general, the value of the parameter is derived as described in Newman and Sproull¹, using the following equations of the line: $X = X1 + (X2 - X1)u$; $Y = Y1 + (Y2 - Y1)u$; $Z = Z1 + (Z2 - Z1)u$. These equations are each inserted into the corresponding plane equation. In the current example, $N = Z1 + (Z2 - Z1)t$.

Since N is between $Z1$ and $Z2$, t is always positive, and the signs of $Z1 - N$ and $Z2 - N$ are used to determine which end to clip. If $Z1 - N$ is negative, the $P1$ end is clipped, using the value of t to determine the delta in $X1$ and $Y1$. The coordinates for the new endpoint of the shortened line segment are given by:

$$X1' = X1 + (X2 - X1) \times t, Y1' = Y1 + (Y2 - Y1) \times t, Z1' = N.$$

¹Newman, W. M., and Sproull, R. F., *Principles of Interactive Computer Graphics*, McGraw-Hill, 1979.

Similarly for the case when the P2 end must be clipped:

$$X2' = X1 + (X2 - X1) \times t, Y2' = Y1 + (Y2 - Y1) \times t, Z2' = N.$$

An alternative to clipping to one plane at a time entails clipping to all six planes at once. Both approaches are examined in the following sections.

Clipping to One Plane at a Time

When a pipeline of six processors is used, each clipping the same line to one plane, each processor must wait for data from the previous processor and hold its solution until the next processor is ready to receive it. There is no reason to seek shortcuts through the computations by including branches in the program because there is little point in one of the processors completing its task earlier than the rest. This statement is true whether the six processors are driven from the same or from separate sequencers. Similarly, operating the pipeline asynchronously buys little time. Synchronous operation in the case of a clipping pipeline is likely to be almost as fast as, and much simpler and cheaper than, asynchronous operation.

Because shortcuts are not beneficial, the program can be written assuming the maximum amount of work will be required at each stage, whether the line requires clipping at that stage or not. If it is assumed that invisible lines are caught and eliminated as a separate, initial computation, branches from the clipping pipeline can be eliminated entirely. An alternative approach, in which branches would be beneficial, involves using two, three, or more 'ACT8847 chips in parallel, rather than as a pipeline, each performing all six stages of clipping for individual lines. The program lends itself to this approach because the computations in each stage of the clipping pipeline are identical.

The method for clipping a line segment against the $Z = N$ plane as one stage in a clipping pipeline, assuming invisible lines have been previously eliminated, will be illustrated. Two t values are computed t_1 for clipping the P1 end of the line segment and t_2 for clipping the P2 end. If $Z1 < N$, $t_1 = (Z1 - N)/(Z1 - Z2)$; otherwise, $t_1 = 0$. If $Z2 < N$, $t_2 = (Z2 - N)/(Z1 - Z2)$; otherwise, $t_2 = 0$. The new endpoints for the line segment are computed as follows:

$$\begin{aligned} X1' &= X1 + (X2 - X1) \times t_1, \\ Y1' &= Y1 + (Y2 - Y1) \times t_1, \\ Z1' &= Z1 + (Z2 - Z1) \times t_1 \end{aligned}$$

$$\begin{aligned} X2' &= X2 - (X2 - X1) \times t_2, \\ Y2' &= Y2 - (Y2 - Y1) \times t_2, \\ Z2' &= Z2 - (Z2 - Z1) \times t_2. \end{aligned}$$

Note that the denominator is the same in the equations for t_1 and t_2 ; it is this reciprocal computation that is expensive in time. In the 'ACT8847, the built-in divide is very fast.

t_1 and t_2 are calculated by step 18, and the entire operation completes in step 27. The data flow is shown in Table 31. A six-processor 'ACT8847 system operating at 30 MHz would clip 1.2 million line segments per second with a new operation beginning every 25 cycles.

Table 31. Data Flow for Clipping a Line Segment Against the Z = N Plane Using the SN74ACT8847

| | | | | | | | | | | | | | | | |
|--------|----|----|----|----|----|---------|-----|-----|----|-----|---------|----------------------|----|---------------------------|--|
| RA | Z1 | Z1 | Z2 | X2 | | | | 0.5 | Y2 | H1' | H2' | SAME AS FOR '8837 | | | |
| RB | Z2 | N | N | X1 | | | | d | | | | | | | |
| S | | | d | H1 | H2 | X2 - X1 | H1' | H2' | | | Y2 - Y1 | | | | |
| P | | | | | | | | | | 1/D | | t1 | t2 | STEPS 20 THRU 28 | |
| C | | | | | H1 | H2 | | | | | 1/D | | t1 | | |
| Y | | | d | | | X2 - X1 | H1' | H2' | | | Y2 - Y1 | | | | |
| STATUS | | | | | | | | | | | | | | | |
| CLK | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 14 | 15 | 16 | 17 | 18 | | |

Since the performance levels obtained from the six-chip systems described below are slower than the rate of endpoint transformation by a single-chip system, some further speed improvement is desirable. Hence, rather than going through the code for clipping to the X and Y planes, another approach is proposed.

Clipping to All Six Planes at a Time

The "window edge clipping method" derived in Newman and Sproull can be used to clip to all six planes at once. Recall that the viewing volume for a perspective view is a pyramid defined by the following plane equations:

$$X = K \times Z, X = -K \times Z, Y = K \times Z, Y = -K \times Z, Z = N, Z = F,$$

where $K = S/N$, as defined in a previous section. Given a segment with endpoints $P1 = (X1, Y1, Z1)$ and $P2 = (X2, Y2, Z2)$, to perform the entire clipping operation on all six planes at once, the following two six-tuples must be computed:

$$Q = (W1 + X1, W1 - X1, W1 + Y1, W1 - Y1, Z1 - N, F - Z1) = (Q1, Q2, \dots),$$

$$R = (W2 + X2, W2 - X2, W2 + Y2, W2 - Y2, Z2 - N, F - Z2) = (R1, R2, \dots),$$

where $W1 = KZ1$ and $W2 = KZ2$.

Consider the case where $X1 < W1$. Then, $W1 + X1 < 0$; i.e., $Q1 < 0$. In general, a negative element of Q indicates that $P1$ is on the invisible side of one of the clipping planes, while a negative element of R indicates the same for $P2$. To clip the line, the six parameters t_i for clipping the $P1$ end and the six parameters s_j for clipping the $P2$ end are computed. Here, $t_i 20 = 20Q_i / (Q_i - R_i)$ and $s_j = R_j / (R_j - Q_j)$. (Again, the equations of the line as described in Newman and Sproull are used).

For example, to find the value t_1 for clipping P1 to the $X = -W = -KZ$ plane, the following equation is used:

$$X_1 + (X_2 - X_1)t_1 = -K[Z_1 + (Z_2 - Z_1)t_1].$$

Solving for t_1 ,

$$t_1 = (X_1 + W_1)/[(X_1 + W_1) - (X_2 + W_2)] = Q_1 / (Q_1 - R_1).$$

In general, $t_i = Q_i/(Q_i - R_i)$. Similarly, $s_i = R_i/(R_i - Q_i)$.

To actually carry out the computations of t_i and s_i , the trick discussed above is performed, and each element of Q and R is replaced with the difference of the element and its absolute value, to form Q' and R' . That is,

$$Q'_i = 2 \times Q_i \text{ if } Q_i < 0, \text{ and } Q'_i = 0 \text{ otherwise.}$$

$$R'_i = 2 \times R_i \text{ if } R_i < 0, \text{ and } R'_i = 0 \text{ otherwise.}$$

Next calculated is $t_i = Q'_i/[2(Q_i - R_i)]$ and $s_i = R'_i/[2(R_i - Q_i)]$, followed by $T_1 = \text{MAX}(t_i)$ and $T_2 = 1 - \text{MAX}(s_i)$. The P1 end is clipped using T_1 and the P2 end is clipped using T_2 .

In an 'ACT8847 three-processor parallel system, in which each processor is given the task of computing two t_i and two s_i values, computing the Q'_i and R'_i values takes 14 cycles, with the values of $Q_i - R_i$ computed by step 13. The six divides, $0.5/(Q_i - R_i)$, are completed in step 30, assuming an 8-bit seed ROM is used. The max/min operations take place in parallel in two processors and complete at step 54 (24 + 30), and the new endpoints are ready by step 60 (6 + 54).

The data flow and program listing for computing t_1, t_2, s_1 , and s_2 by one of the three 'ACT8847 processors is given in Tables 32 and 33.

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|

NOTE: Cycles 13, 16, 17, 19, ... 28 compute $1/D1 = 0.5/d1$; Cycles 14, 16, 18, 20, ... 28 compute $1/D2 = 0.5/d2$; $d1 = Q1 - R1$, $d2 = Q2 - R2$.

Table 32. Data Flow for Computing t_1 , t_2 , s_1 , and s_2 Using an SN74ACT8847

| | | | | | | | | | | |
|--------|--------|---------|---------|-----|--------|------|------|------|-----|-----|
| CHAIN | Y | Y | Y | Y | Y | N | N | N | Y | Y |
| RA | K | K | | | | | | | W2 | Q1 |
| RB | Z1 | Z2 | X1 | X1 | X2 | | | | X2 | R1 |
| S | | | | | Q1 | Q2 | R1 | Q1' | Q2' | R1' |
| P | | | W1 | W2 | | | | | | |
| C | | | | W1 | W2 | Q1 | Q2 | R1 | | |
| Y | | | | | | | | Q1' | Q2' | R1' |
| STATUS | | | | | | | | | | |
| CLK | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| CHAIN | Y | N | Y | Y | Y | Y | Y | Y | Y | Y |
| RA | Q2 | | R0 | | | | | | | |
| RB | R2 | | d1 | | | | | | | |
| S | R2 | Q1 - R1 | Q2 - R2 | R2' | R0 | | T0 | | | |
| P | | | | | d × R0 | | R0 | | R1 | |
| C | | R2 | | | | | | | | |
| Y | | Q1 - R1 | Q2 - R2 | R2' | | | | | | |
| STATUS | | | | | | | | | | |
| CLK | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| CHAIN | Y | Y | Y | Y | Y | Y | Y | Y | N | N |
| RA | | | | | Q1' | Q2' | R1' | | | |
| RB | O - S | | | | | | | R2' | | |
| S | R1 | | T1 | | | | | 1/D2 | | |
| P | d × R1 | | O - SR1 | | 1/D1 | 1/D2 | t1 | t2 | S1 | S2 |
| C | | | | | | 1/D1 | 1/D1 | | | |
| Y | | | | | | | t1 | t2 | S1 | S2 |
| STATUS | | | | | | | | | | |
| CLK | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |

NOTE: Cycles 13, 15, 17, 19, . . . , 25 compute $1/D1 = 0.5/d1$; Cycles 14, 16, 18, 20, . . . , 26 compute $1/D2 = 0.5/d2$, $d_i = Q_i - R_i$.

Table 33. Program Listing for Three-Processor Clip to Compute t_1 , t_2 , s_1 , and s_2 Only

| | REGISTER TRANSFERS | | ALU OPERATION | MULTIPLIER OPERATION |
|------------------------|--------------------|---------|---------------|----------------------|
| 1. | LOAD RA, RB | | | MULT (RA,RB) |
| 2. | LOAD RA, RB | | | MULT (RA,RB) |
| 3. | LOAD RB | C←P | ADD (P,RB) | |
| 4. | LOAD RB | C←P | ADD (C, -RB) | |
| 5. | LOAD RB | C←S | ADD (C,RB) | |
| 6. | | Y←S C←S | ADD (C, - C) | |
| 7. | | Y←S C←S | ADD (C, - C) | |
| 8. | | Y←S | ADD (C, - C) | |
| 9. | LOAD RA, RB | | ADD (RA, -RB) | |
| 10. | LOAD RA, RB | Y←S | ADD (RA, -RB) | |
| 11. | LOAD RB, RB | Y←S C←S | ADD (RA, -RB) | |
| 12. | | Y←S | ADD (C, - C) | |
| CODE FOR TWO DIVISIONS | | | | |
| 25. | LOAD RA | Y←S C←P | | MULT (RA,P) |
| 26. | LOAD RA | Y←S | ADD (P,O) | MULT (RA,P) |
| 27. | | Y←S | | MULT (RA,C) |
| 28. | | Y←S | | MULT (S,RB) |

This approach facilitates the transform of 576,000 line segments in a 3-chip 'ACT8847 system running at 30 MHz. If branches are permitted in the sequencer, a considerable speedup is available for situations in which a large proportion of line segments are either invisible, and may be eliminated, or are completely visible, and may be passed without clipping. A single-processor system takes no more than 32 cycles, sometimes as few as 10 cycles, to reject an invisible line, whereas it takes 91 cycles to process lines that need both ends clipped. Hence, in a situation where 50% of the line segments are invisible, the speed is in excess of 540,000 segments/second at 30 MHz. It is not uncommon for 80% of lines to be invisible, in which case, the speed would increase to 877,000 line segments at 30 MHz.

To take advantage of this speedup, the only change in the sequence given above is that while computing Q and R, the logical AND and OR is formed for the signs of the corresponding pairs of values, Q_i and R_i .

Table 34. $A > B$ Comparison Function Table

| Sign Q_i | Sign R_i | Sign $A = -Q_i \times R_i $ | Sign $B = Q_i \times R_i$ | $A > B$ | $A = B$ |
|------------|------------|------------------------------|-----------------------------|---------|---------|
| - | - | + | - | T | F |
| - | + | + | + | F | T |
| + | - | - | - | F | T |
| + | + | - | + | F | F |

The $A > B$ status provides the needed AND function of the sign bits of Q_i and R_i . In computing these $A > B$ values, if $A > B$ is TRUE, the sequencer branches to code that rejects the line as invisible. A comparison $A > B$ of $A = (Q_i \times |R_i|)$ and $B = (|Q_i| \times R_i)$ gives the logical AND of the complement of the sign bits. It is TRUE when both Q_i and R_i are positive. If all six values are TRUE, the sequencer can branch to code that passes the line segment unclipped.

For a three-processor parallel system, lockstep operation with a single sequencer is still possible since all three processors are working on the same line segment, and the branch options apply equally to them all. The estimated time for a three-processor system is 56 cycles.

Now that the operations have been reduced to a minimum, the remaining steps are necessarily sequential. Rejecting invisible or passing totally visible line segments without division, however, is still beneficial.

Clipping in the Screen System

In most graphics systems, full line clipping is not performed in the eye system. Instead, a trivial accept/reject test is performed, in which the line segments are simply tested against the six clipping planes. If a line has both ends on the invisible side of any one of the clipping planes, it is rejected. Lines surviving this test may still be outside the viewing pyramid. In any case, the lines are transformed to the screen coordinate system and then clipped against a cube defined by the simple plane equations $-1 < (X, Y, Z) < 1$. The next three sections describe this process.

Trivial Accept/Reject Test

In the eye system, the clipping planes are:

$$X = W, X = -W, Y = W, Y = -W, Z = N, \text{ and } Z = F,$$

where $W = K \times Z$. After $-W1$ and $-W2$ are computed, a sequence of comparison operations are performed, summarized as follows:

with $X1$ in RB and $-W1$ in P, $P > RB$ (i.e., $-W1 > X1$)
 with $X1$ in RA and $-W1$ in C, $RA > |C|$ (i.e., $X1 > W1$)
 with $Y1$ in RB and $-W1$ in C, $C > RB$
 with $Y1$ in RA, $RA > |C|$ comparison
 with $Z1$ in RB and n in RA, $RA > RB$ (i.e. $N > Z1$)
 with $Z1$ in RA and F in RB, $RA > RB$ (i.e. $Z1 > F$).

These six operations are carried out in successive cycles and then repeated for $(X2, Y2, Z2)$. The two six-tuples are saved off-chip and a bit-wise AND is carried out. If any one of the resulting six boolean values is TRUE, the line is rejected. This entire operation takes only 16 cycles, thereby providing a speed of 2,143,000 line segments per second at 30 MHz. The data flow for an accept/reject test is given in Table 35. Accept/reject testing of individual points takes only 8 cycles.

Table 35. Data Flow for Accept/Reject Testing

| CHAIN | N | N | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | N | N |
|--------|----|----|-----|------------|-------|------------|-------|------|------|------------|-------|------------|-------|------|------|----|---|
| RA | K | K | | X1 | | Y1 | N | Z1 | −W2 | X2 | −W2 | Y1 | N | Z2 | | | |
| RB | Z1 | Z2 | X1 | | Y1 | | Z1 | F | X2 | −W2 | Y1 | −W2 | Z2 | F | | | |
| S | | | | | | | | | | | | | | | | | |
| P | | | −W1 | −W2 | | | | | | | | | | | | | |
| C | | | −W1 | −W1 | −W1 | −W1 | | | | | | | | | | | |
| Y | | | −W2 | | | | | | | | | | | | | | |
| STATUS | | | | −W1 >X1 | X1>W1 | −W1 >Y1 | Y1>W1 | N>Z1 | Z1>F | −W2 >X2 | X2>W2 | −W2 >Y2 | Y2>W2 | N>Z2 | Z2>F | | |
| CLK | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | |

Transformation to the Screen System

After the line segments have passed the trivial accept/reject test, they are transformed to the screen coordinate system. The following transformation is first applied to the Z coordinate in order to scale its clipping planes to $Z' = -W$, and $Z' = W$:

$$Z' = [-W \times (F + N)] / (F - N) + (2 \times W \times Z) / (F - N).$$

The value of $1/(F - N)$ is constant for all line segments and is therefore computed only once. In fact, two constants, $a = 2K/(F - N)$ and $b = (F + N)/2$, can be available so that $Z' = Z \times a \times (b + Z)$. (Note that other transformations on Z can also be used.)

After the trivial accept/reject test, the following transformation to the screen system occurs:

$$X_s = X/W, Y_s = Y/W, Z_s = Z'/W.$$

The clipping planes then have these equations:

$$X_s = -1, X_s = 1, Y_s = -1, Y_s = 1, Z_s = -1, Z_s = 1.$$

Z1' and Z2' can be formed in 8 cycles. Only two reciprocals, 1/W1 and 1/W2, need to be computed, and they can be interleaved and completed in 12 cycles in an '8847. The line segment is transformed to the screen system in a further 6 cycles. The total is 26 cycles for the 'ACT8847. A single-processor system would transform 1.2 million line segments per second at 30 MHz.

Note that the above projection does not preserve planarity. See Newman and Sproull for perspective projections that do preserve planes.

The Clipping Operation

The final operation on line segments is to clip them to the cube:

$$X_s = 1, X_s = -1, Y_s = 1, Y_s = -1, Z_s = 1 \text{ and } Z_s = -1.$$

It is important to realize that the required resolution of X_s , Y_s and Z_s may only be 10 or 11 bits.

Consider a three-processor pipeline, with each processor clipping against two parallel planes. The first will clip against the x planes $-1 < X < 1$. For clipping the P1 end of the line segment, $Q = (1 + X1, 1 - X1)$ is computed and Q' is formed, where $Q_i' = Q_i - |Q_i|$. I.e.,

$$Q_1' = 2(1 + X1), \text{ if } (1 + X1) < 0; Q_1' = 0 \text{ otherwise.}$$

$$Q_2' = 2(1 - X1), \text{ if } (1 - X1) < 0; Q_2' = 0 \text{ otherwise.}$$

At least one of Q_i' will be zero; the other will be negative. Hence, $\text{MIN}(Q_1', Q_2) = Q_1' + Q_2' = [(1 + X1) - |1 + X1|] + [(1 - X1) - |1 - X1|]$. Therefore, $\text{MIN}(Q_1', Q_2') = (1 - |X1|) - |1 - |X1||$. So, $t = (m_1 - |m_1|) / 2d$ and $s = (m_2 - |m_2|) / 2d$, where $m_i = 1 - |X_i|$ and $d = X1 - X2$. Note that only one reciprocal is required per processor.

A three-processor parallel system would have each processor work on one dimension, supplying its pair of max parameters to a "second stage." The second stage would receive (t_x, s_x) , (t_y, s_y) , (t_z, s_z) from the above system, compute $\max(t) = T$ and $\max(s) = S$, and then clip line as before:

$$X1' = X1 + (X2 - X1)T,$$

$$X2' = X2 - (X2 - X1)S.$$

The data flow and program listing for the program run by a processor working on the X dimension are given in Tables 36 and 37.

Table 36. Data Flow for the X Processor

| CHAIN | Y | N | Y | Y | N | Y | N | Y | Y | Y | Y | Y | Y | Y |
|--------|----|-----|-----|----|----|--------|----|----|----|----|----|--------|----|-------|
| RA | X1 | | | R0 | | | | | | d | | | | |
| RB | X2 | X1 | X2 | d | | | | | | | | 0.5 | | |
| S | | | d | m1 | m2 | R0 | n1 | T0 | n2 | | | R1 | | T1 |
| P | | | | | | d × R0 | | R0 | | R1 | | d × R1 | | 0.5R1 |
| C | | | | | m1 | m2 | m2 | | | | | | | |
| Y | | | d | | | | n1 | | n2 | | | | | |
| STATUS | | | | | | | | | | | | | | |
| CLK | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| CHAIN | Y | N | N | Y | Y | | | | | | | | | |
| RA | | n1 | n2 | | | | | | | | | | | |
| RB | | | | | | | | | | | | | | |
| S | | | | | | | | | | | | | | |
| P | | 1/D | | t | s | | | | | | | | | |
| C | | | 1/D | | | | | | | | | | | |
| Y | | | | t | s | | | | | | | | | |
| STATUS | | | | | | | | | | | | | | |
| CLK | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 |

NOTE: $d = X1 - X2$; $n_i = m_i - |m_i|$

Table 37. Program Listing for the X Processor

| REGISTER TRANSFERS | ALU OPERATION | MULTIPLIER OPERATION |
|--------------------|----------------|----------------------|
| 1. LOAD RA, RB Y←S | ADD (RA, -RB) | |
| 2. LOAD RA, RB | ADD (RA, -RB) | |
| 3. LOAD RA, RB | ADD (RA, -RB) | |
| 4. LOAD RA, RB | ADD (RA, O) | MULT (RA, RB) |
| 5. Y←S | ADD (C, - C) | |
| 6. | ADD (2, -P) | MULT (S, 1) |
| 7. Y←S | ADD (C, - C) | |
| 8. | | MULT (S, P) |
| 9. | | |
| 10. LOAD RA | ADD (P, O) | MULT (RA, P) |
| 11. | | |
| 12. LOAD RB | ADD (2, -P) | MULT (S, RB) |
| 13. | | |
| 14. | | MULT (S, P) |
| 15. | | |
| 16. LOAD RA Y←P | | MULT (RA, P) |
| 17. LOAD RA Y←P | | MULT (RA, P) |
| 18. | | |
| 19. | | |

The three-processor parallel clipping system operates on a fixed loop of 17 instructions and can therefore clip 1.76 million line segments per segment at 30 MHz. The second stage could not keep up with this rate without being implemented as several processors. A single processor can form the two max values in 23 cycles (a loop of 21 cycles) while two processors would take only 12 cycles (a loop of 10). The final clipping of the two endpoints takes about 11 cycles (a loop of 9 cycles).

To summarize, the fastest clipping system operates in the normalized screen coordinate system. It has six processors arranged in three stages — a three-processor parallel system with each processor working on each dimension and a single-processor third stage to clip the endpoints. The combined speed would be equal to that of the first stage, as previously described. A slightly slower four-processor system would use one processor for computing the two max values in the second stage.

Summary of Graphics Systems Performance

The previous section considered several approaches to the design of computer graphics systems based on the 'ACT8847. Table 38 summarizes the results. Table 39 shows the options available in combining the sub-systems listed in Table 38 into a design for a graphics system.

Table 38. Summary of Graphics Systems Performance

| SUB-SYSTEM | | SPEED AT 30 MHz |
|-----------------------------------|-------------------|------------------|
| a Transform, 4×4 matrix, | 1 'ACT8847 cycle | 1.875 M points/s |
| b Transform, 3×3 matrix, | 1 'ACT8847 cycle | 2.5 M points/s |
| c Eye clipping pipe, | 6 'ACT8847 cycles | 1.2 M lines/s |
| d Eye clipping | 3 'ACT8847 cycles | 0.576 M lines/s |
| e Eye Accept/Reject test | 1 'ACT8847 cycle | 2.143 M lines/s |
| f Screen clipping | 5 'ACT8847 cycles | 1.76 M lines/s |
| g Screen clipping | 4 'ACT8847 cycles | 1.42 M lines/s |

Table 39. Available Options for Graphics System Designs

| SUB-SYSTEM | | SPEED AT 30 MHz |
|---|-------------------|-----------------|
| I (a or b) + c, | 7 'ACT8847 cycles | 1.2 M lines/s |
| II (a or b) + d, | 2 'ACT8847 cycles | 0.576 M lines/s |
| III (a or b) + f, | 6 'ACT8847 cycles | 1.76 M lines/s |
| IV $2 \times (a \text{ or } b) + c + g$, | 7 'ACT8847 cycles | 3.75 M lines/s |

In the fourth system, it is assumed that 2 processors are used for the transform of endpoints so as to balance the high clipping rate. It is also assumed that the accept/reject stage will eliminate more than 60% of the line segments so that the clipping system can keep up with the transform processors.

Summary of Graphics Systems Performance

The previous section considered several approaches to the design of computer graphics systems based on the ACT8847. Table 38 summarizes the results. Table 39 shows the options available in combining the sub-systems listed in Table 38 into a design for a graphics system.

Table 38. Summary of Graphics Systems Performance

| SUB-SYSTEM | | SPEED AT 30 MHz |
|---------------------------|------------------|------------------|
| a Transform, 4 x 4 matrix | 1 ACT8847 cycle | 1.875 M points/s |
| b Transform, 3 x 3 matrix | 1 ACT8847 cycle | 2.5 M points/s |
| c Eye clipping pipe | 8 ACT8847 cycles | 1.2 M lines/s |
| d Eye clipping | 3 ACT8847 cycles | 0.875 M lines/s |
| e Eye Accept/Reject test | 1 ACT8847 cycle | 2.143 M lines/s |
| f Screen clipping | 5 ACT8847 cycles | 1.76 M lines/s |
| g Screen clipping | 4 ACT8847 cycles | 1.42 M lines/s |

Table 39. Available Options for Graphics System Designs

| SUB-SYSTEM | | SPEED AT 30 MHz |
|-------------------------|------------------|-----------------|
| I (a or b) + c | 7 ACT8847 cycles | 1.2 M lines/s |
| II (a or b) + d | 2 ACT8847 cycles | 0.875 M lines/s |
| III (a or b) + f | 6 ACT8847 cycles | 1.76 M lines/s |
| IV 2 x (a or b) + c + g | 7 ACT8847 cycles | 2.76 M lines/s |

In the fourth system, it is assumed that 2 processors are used for the transform of endpoints so as to balance the high clipping rate. It is also assumed that the accept/reject stage will eliminate more than 60% of the line segments so that the clipping system can keep up with the transform processors.

| | |
|---|-----------|
| General Information | 1 |
| SN74ACT8818A 16-Bit Microsequencer | 2 |
| SN74ACT8832A 32-Bit Registered ALU | 3 |
| SN74ACT8836A 32- × 32-Bit Parallel Multiplier | 4 |
| SN74ACT8841A Digital Crossbar Switch | 5 |
| SN74ACT8847 64-Bit Floating-Point/Integer Unit | 6 |
| SN74ACT8847 Application Information | 7 |
| SN74ACT8867 32-Bit Vector Processor Unit | 8 |
| Design Support | 9 |
| Mechanical Data | 10 |

1 General Information

2 SN74ACT8310A 10-Bit Monostable Multivibrator

3 SN74ACT8832A 32-Bit Registered with

4 SN74ACT8838A 32 × 32-Bit Parallel Multiplier

5 SN74ACT8841A Digital Crossbar Switch

6 SN74ACT8847 84-Bit Floating-Point/Integer Unit

7 SN74ACT8847 Application Information

8 SN74ACT8887 32-Bit Vector Processor Unit

9 Design Support

10 Mechanical Data

- **Fast Numeric Processor Optimized for Graphics**
- **Simultaneous Independent Operation of Multiplier and ALU**
- **Onboard Six-Port Register File**
– 46 Words X 40 Bits
- **Operates in Pipelined or Flow-Through Mode**
- **62-ns Cycle Time in Fully Pipelined Mode**
- **On-Board Floating-Point Seed ROM for Newton-Raphson Division and Square Root**
- **Three 32-Bit Ports Support High-Data Bandwidth**
- **Selectable Format Conversion From Fixed-to Floating-Point at Input Ports**
- **Selectable Format Conversion From Floating- to Fixed-Point at Output Ports**
- **Low-Power 1- μ m EPIC™ CMOS Process**

description

On a single chip the SN74ACT8867 Vector Processor Unit (VPU) combines a multiplier, Arithmetic Logic Unit (ALU), and a six-port register file for floating-point and integer arithmetic. The VPU performs single-precision floating-point operations and fixed-point operations in single or double precision.

Applications such as graphics workstations and high-end digital signal processors can make use of the speed and flexibility offered by the independent multiplier and ALU, which are separately programmable and operate concurrently. The register file can be double-pumped to input four operands within one clock period, matching the input bandwidth of the register file to the data bandwidth through the multiplier and ALU in concurrent operation.

The 'ACT8867 accepts operands from its two 32-bit input buses as 2s complement integers, 2s complement fractions, or floating-point numbers in Digital Equipment Corporation (DEC) 'F' format. Internally, the VPU supports 40-bit floating-point arithmetic and 32-bit or 64-bit fixed-point arithmetic.

The 'ACT8867 has two 32-bit input data buses and a 32-bit output data bus (see functional block diagram). The device supports 40-bit floating-point internally and also supports 2s complement integer operations. Forty-six general-purpose registers allow intermediate results to be stored internally. Two other pseudo-register locations, when addressed, serve as a flow-through function in the register file. In this mode, the register file is bypassed, allowing data to be fed directly from an external source to the processing units.

Format conversion logic is available to translate fixed-point inputs to floating-point format without tying up the ALU. Similarly, floating-point results can be translated to fixed-point outputs by using the logic elements provided for float-to-fixed conversion and binary-point adjustment.

When all the main functional blocks are active simultaneously, the VPU can perform a multiply, an ALU operation, and three format conversions (two fixed-to-floating-point and one floating-to-fixed-point) within a single clock cycle.

The instruction set for the 'ACT8867 is partitioned so that the two processing units, the ALU processor and the multiplier processor, can be operated independently or in parallel.

Single-precision floating-point Newton-Raphson division and square root are supported, along with limited support for double-precision arithmetic. A single-precision seed table for Newton-Raphson division and square root is provided (see Table 6). Appropriate status is generated for all instructions and data types. The VPU can be operated in flow-through mode, or pipeline registers are available to allow operation in pipeline mode to increase system speed. All pipeline registers may be placed in a scan mode for device-level and board-level diagnostics.

The device operates with a 62-ns system clock and is designed with five percent power supply tolerance requirements over the commercial temperature range (0°C to 70°C).

EPIC is a trademark of Texas Instruments Incorporated.

PRODUCTION DATA documents contain information current as of publication date. Products conform to specifications per the terms of Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.



POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

Copyright © 1990, Texas Instruments Incorporated

SN74ACT8867 32-BIT VECTOR PROCESSOR UNIT

description (continued)

Several naming conventions are used throughout this document. All signals are active high unless otherwise noted. Signal transition characteristics are standard TTL levels with positive logic. Bus labels are given with the Least Significant Bit (LSB) designated as bit 0. The Most Significant Bit (MSB) is designated as bit 39 or as appropriate for the bus width. In figures, the MSB is located at the left and the LSB is at the right. The register file is commonly referred to by the abbreviation RF and the multiplier is abbreviated as MULT.

data flow

The ACT8867 VPU is a three-port device with independent multiplier and ALU blocks (see functional block diagram). Input and output functions are enhanced by data format conversion blocks which may operate in parallel with the arithmetic blocks.

Each input data bus is fed to a fixed-to-float conversion block. These blocks contain registers to latch input data, if desired. These blocks can either pass data or convert integer or fractional values to floating-point format. The output of each conversion block is connected to a two-input multiplexer which selects data for the write buses of the register file.

The register file has 46 locations, each of which is 40 bits wide. Two data operands may be written simultaneously through the two write inputs. Four read outputs are available to read any data value and output it to the inputs of the multiplier and ALU.

Both inputs to the multiplier may receive data from the register file, the ALU, or feedback from itself. The A input may also select output from the seed ROM to start division or square-root operations. Data registers are provided on each input to the multiplier and the instruction bits are also registered. A single write enable and two flow-through pins control the function of these registers. The multiplier instructions support single-precision floating-point, integer, and multiple-precision fixed-point operations. Internal output feedback may be registered or flow-through for output from the device.

Both inputs to the ALU may receive data from the register file or feedback. The A input allows feedback from the multiplier and the B input allows feedback from the ALU. Data registers are provided on each input to the ALU and the instruction bits are also registered. A single write enable and two flow-through pins control the function of these registers. The ALU instructions support single-precision floating-point, integer, multiple-precision fixed-point, and logical operations. Output feedback may be registered or flow-through for output from the device.

Results from the multiplier and the ALU may be passed from their output registers to the float-to-fixed conversion block. A multiplexer selects which unit supplies input to this block. This block can convert floating-point numbers to fixed-point values if the format conversion is selected. Rounding may also be optionally performed in this block. Parity is generated from the output of this block.

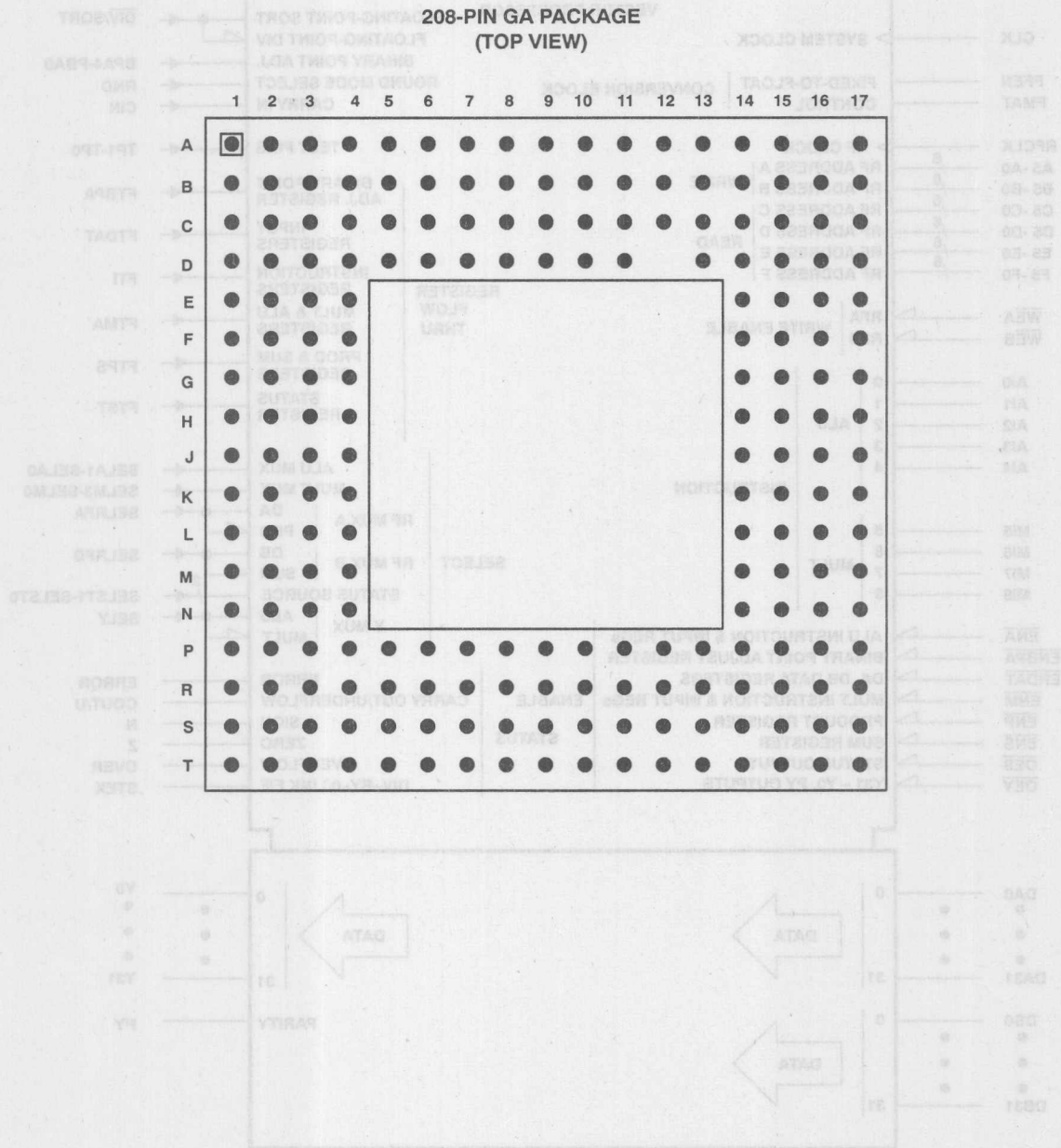
Status is generated from the fixed-to-float, multiplier, ALU, and float-to-fixed blocks and may be latched in the status register. Portions of this status may be selected for output on the six status pins as needed.

Status pins, Y output pins, and parity pins are all buffered for output. The buffers can set the output pins to a high-impedance state.



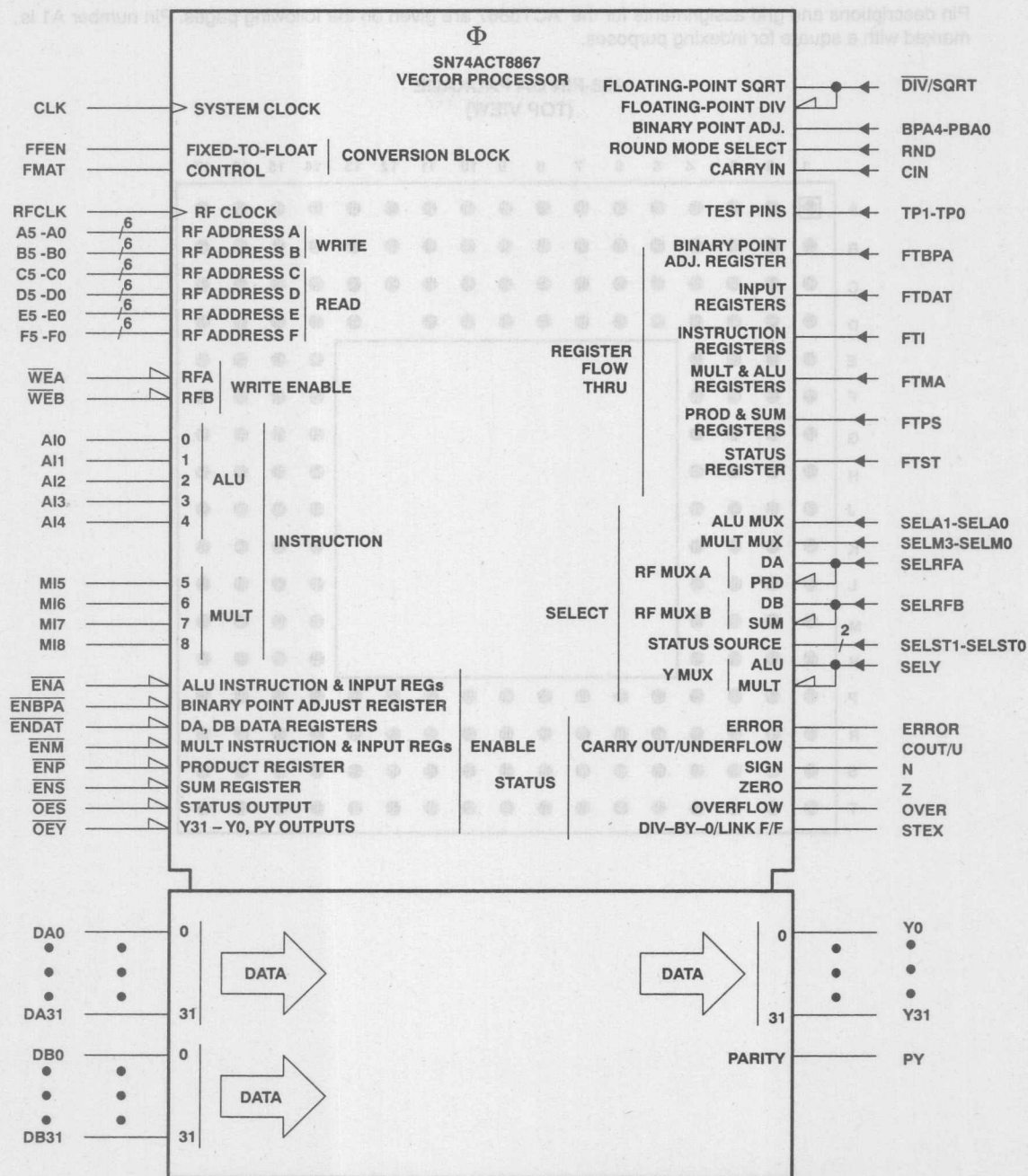
'ACT8867 package information

Pin descriptions and grid assignments for the 'ACT8867 are given on the following pages. Pin number A1 is marked with a square for indexing purposes.



SN74ACT8867 **32-BIT VECTOR PROCESSOR UNIT**

logic symbol†

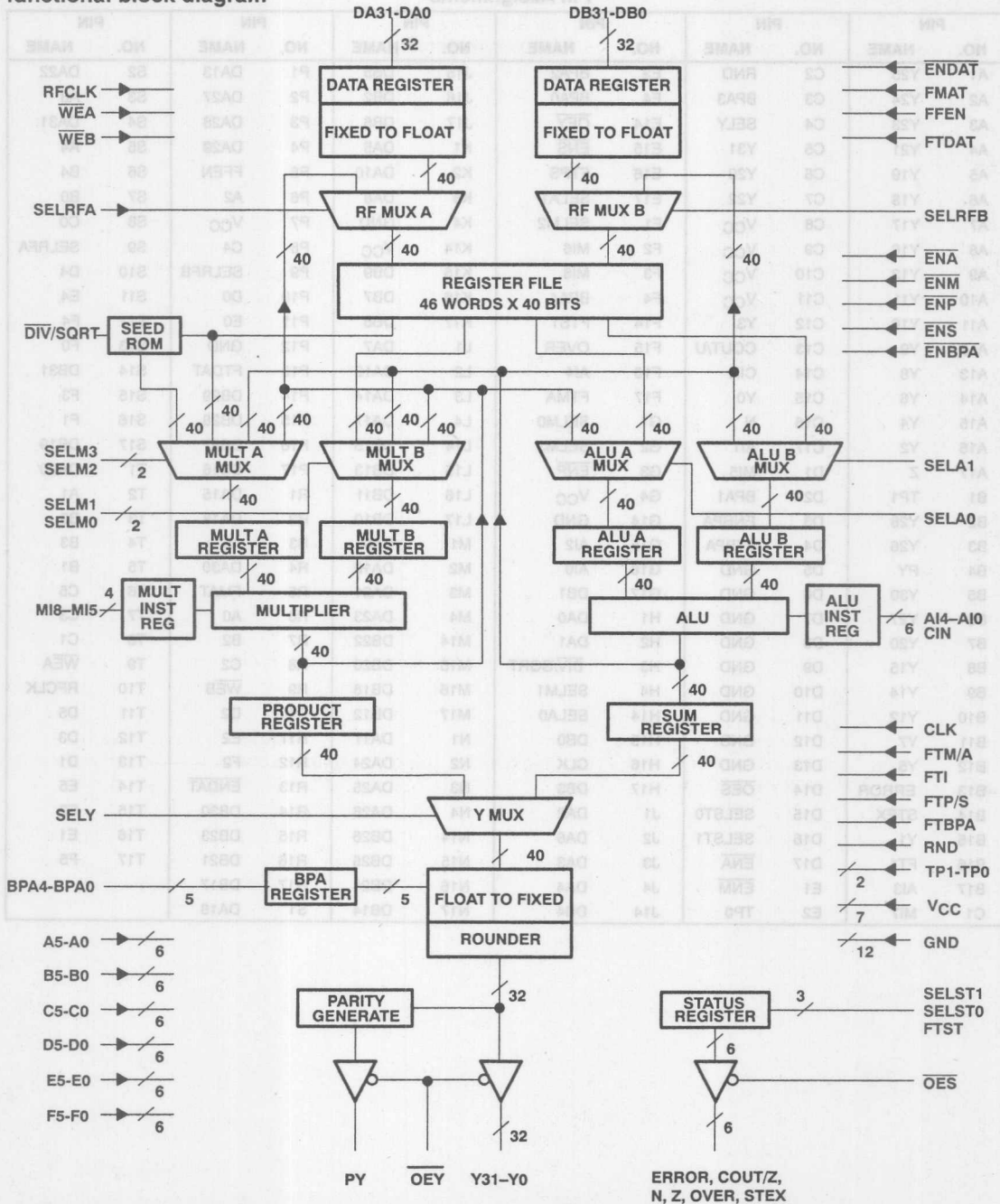


† This symbol is in accordance with ANSI/IEEE Std. 91-1984.

TEXAS
INSTRUMENTS

POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

functional block diagram



32-BIT VECTOR PROCESSOR UNIT

SN74ACT8867
32-BIT VECTOR PROCESSOR UNIT

Terminal Functions

| PIN | | I/O | DESCRIPTION |
|--------------------------------------|--|-----|--|
| NAME | NO. | | |
| A0 A1 A2 A3 A4 A5 | R6 T2 P6 S3 S5 R3 | I | Register File Write Address A. These bits select destination registers for feedback data from the multiplier or input data from the DA bus. Values may be stored in locations 0 to 45. Also, these bits are used with the WEB pin to enable writing. Writing to a nonexistent location performs no useful operation, generates no status error, and does not affect other data in the register file. |
| AI0 AI1 AI2 AI3 AI4 | G16 C17 G15 B17 F16 | I | ALU Instruction. The ALU instruction bits are input through the instruction register in flow-through or registered mode. |
| B0 B1 B2 B3 B4 B5 | S7 T5 R7 T4 S6 T3 | I | Register File Write Address B. These bits select destination registers for feedback data from the multiplier or input data from the DB bus. Values may be stored in locations 0 to 45. Also, these bits are used with the WEB pin to enable writing. Data on the B port is ignored if the B write address matches the A write address. Writing to a nonexistent location performs no useful operation, generates no status error, and does not affect other data in the register file. |
| BPA0 BPA1 BPA2 BPA3 BPA4 | E4 D2 E3 C3 F4 | I | Binary Point Adjust. The binary point adjust pins set the integer size of the fixed-point format used for output float-to-fixed conversions. These bits are input through the BPA register in flow-through or registered mode. |
| C0 C1 C2 C3 C4 C5 | S8 T8 R8 T7 P8 T6 | I | Register File Read Address C. These bits select operands for the A side of the multiplier. The read operation is asynchronous with respect to all other register file controls. |
| CIN | C14 | I | Carry-In Bit Control Line. Carry-in data bit or control line is used in ALU operations. This bit is input through the ALU instruction register in flow-through or registered mode. |
| CLK | H16 | I | Clock. Main system clock for all registers except those associated with the register file and conversion blocks. When enabled, internal registers are latched on the rising edge. |
| COUT/U | C13 | O | Carry-Out/Underflow Status. In unsigned operations, this pin is the carry-out bit and is updated to indicate overflow conditions. When the result of a floating-point operation is too small to be represented in normalized format, this bit is asserted and the result is forced to zero. It is also used in combination with the Z, N, and OVER status bits for compare operations. Additional information under OES. |
| D0 D1 D2 D3 D4 D5 | P10 T13 R10 T12 S10 T11 | I | Register File Read Address D. These bits select operands for the B side of the multiplier. The read operation is asynchronous with respect to all other register file controls. |

SN74ACT8867

32-BIT VECTOR PROCESSOR UNIT

Terminal Functions (Continued)

| PIN | | I/O | DESCRIPTION | PIN | | | |
|------|-----|--|-------------|------|-----|------|-----|
| NAME | NO. | | | NAME | NO. | | |
| DA0 | H1 | | | DA0 | H1 | | |
| DA1 | H2 | | | DA1 | H2 | | |
| DA2 | J1 | | | DA2 | J1 | | |
| DA3 | J3 | | | DA3 | J3 | | |
| DA4 | J4 | | | DA4 | J4 | | |
| DA5 | K1 | | | DA5 | K1 | | |
| DA6 | L2 | | | DA6 | L2 | | |
| DA7 | L1 | | | DA7 | L1 | | |
| DA8 | K3 | | | DA8 | K3 | | |
| DA9 | M1 | | | DA9 | M1 | | |
| DA10 | K2 | | | DA10 | K2 | | |
| DA11 | N1 | | | DA11 | N1 | | |
| DA12 | L2 | Data Bus A Side. Input 32-bit data bus to the A side of the register file. The data format may be converted or passed by the fixed-to-float block. The format is selected by the FFEN and FMAT pins. | | DA12 | L2 | | |
| DA13 | P1 | | | DA13 | P1 | | |
| DA14 | L3 | | | DA14 | L3 | | |
| DA15 | R1 | | | DA15 | R1 | | |
| DA16 | S1 | | | DA16 | S1 | | |
| DA17 | L4 | | | DA17 | L4 | | |
| DA18 | R2 | | | DA18 | R2 | | |
| DA19 | M2 | | | DA19 | M2 | | |
| DA20 | T1 | | | DA20 | T1 | | |
| DA21 | M3 | | | DA21 | M3 | | |
| DA22 | S2 | | | DA22 | S2 | | |
| DA23 | M4 | | | DA23 | M4 | | |
| DA24 | N2 | | | DA24 | N2 | | |
| DA25 | N3 | | | DA25 | N3 | | |
| DA26 | N4 | | | DA26 | N4 | | |
| DA27 | P2 | | | DA27 | P2 | | |
| DA28 | P3 | | | DA28 | P3 | | |
| DA29 | P4 | | | DA29 | P4 | | |
| DA30 | R4 | | | DA30 | R4 | | |
| DA31 | S4 | | | DA31 | S4 | | |
| DB0 | H15 | | | | | DB0 | H15 |
| DB1 | G17 | | | | | DB1 | G17 |
| DB2 | J16 | | | | | DB2 | J16 |
| DB3 | H17 | | | | | DB3 | H17 |
| DB4 | J14 | DB4 | J14 | | | | |
| DB5 | J15 | DB5 | J15 | | | | |
| DB6 | J17 | DB6 | J17 | | | | |
| DB7 | K16 | DB7 | K16 | | | | |
| DB8 | K17 | DB8 | K17 | | | | |
| DB9 | K15 | DB9 | K15 | | | | |
| DB10 | L17 | DB10 | L17 | | | | |
| DB11 | L16 | Data Bus B Side. Input 32-bit data bus to the B side of the register file. The data format may be converted or passed by the fixed-to-float block. The format is selected by the FFEN and FMAT pins. | | | | DB11 | L16 |
| DB12 | M17 | | | DB12 | M17 | | |
| DB13 | L15 | | | DB13 | L15 | | |
| DB14 | N17 | | | DB14 | N17 | | |
| DB15 | L14 | | | DB15 | L14 | | |
| DB16 | P17 | | | DB16 | P17 | | |
| DB17 | R17 | | | DB17 | R17 | | |
| DB18 | M16 | | | DB18 | M16 | | |
| DB19 | S17 | | | DB19 | S17 | | |
| DB20 | M15 | | | DB20 | M15 | | |
| DB21 | R16 | | | DB21 | R16 | | |
| DB22 | M14 | | | DB22 | M14 | | |
| DB23 | R15 | DB23 | R15 | | | | |
| DB24 | N16 | DB24 | N16 | | | | |
| DB25 | N15 | DB25 | N15 | | | | |
| DB26 | N14 | DB26 | N14 | | | | |
| DB27 | P16 | DB27 | P16 | | | | |
| DB28 | P15 | DB28 | P15 | | | | |
| DB29 | P14 | DB29 | P14 | | | | |
| DB30 | R14 | DB30 | R14 | | | | |
| DB31 | S14 | DB31 | S14 | | | | |



Terminal Functions (Continued)

| PIN | | I/O | DESCRIPTION |
|----------------------------------|--|-----|--|
| NAME | NO. | | |
| $\overline{\text{DIV/SQRT}}$ | H3 | I | Divide/Square Root. The divide table from the divide/square root ROM is selected when pin is low and square root table is selected when pin is high. |
| E0 E1 E2 E3 E4 E5 | P11 T16 R11 T15 S11 T14 | I | Register File Read Address E. These bits select operands for the A side of the ALU. The read operation is asynchronous with respect to all other register file controls. |
| $\overline{\text{ENA}}$ | D17 | I | Enable Control for ALU Input Registers and ALU Instruction Register, active low. Data will be latched on the rising edge of CLK when this signal is low. |
| $\overline{\text{ENBPA}}$ | D3 | I | Enable Control for Binary Point Adjust Register, active low. Data will be latched on the rising edge of CLK when this signal is low. |
| $\overline{\text{ENDAT}}$ | R13 | I | Enable Control for the Input Registers and Multiplier Instruction Register, active low. Data will be latched on the rising edge of CLK when this signal is low. |
| $\overline{\text{ENIM}}$ | E1 | I | Enable Control for Multiplier Input Registers and Multiplier Instruction Register, active low. Data will be latched on the rising edge of CLK when this signal is low. |
| $\overline{\text{ENP}}$ | G3 | I | Enable Control for the Product Register, active low. Output from the multiplier will be latched on the rising edge of CLK when this signal is low. |
| $\overline{\text{ENS}}$ | E15 | I | Enable Control for the Sum Register, active low. Output from the ALU will be latched on the rising edge of CLK when this signal is low. |
| ERROR | B13 | O | Chip Error Status. This bit is updated every clock cycle, and is set when an overflow or underflow has occurred in any of the three status sources, a divide by zero is attempted in the multiplier, or a reserved floating-point operand is input to the chip. The ERROR signal works only in conjunction with floating-point and 2s complement operations in the arithmetic blocks. This pin will not be set by the ALU during a floating-point compare operation and should be carefully interpreted for multiple precision operations. Additional information under OES. |
| F0 F1 F2 F3 F4 F5 | S13 S16 R12 S15 S12 T17 | I | Register File Read Address F. These bits select operands for the B side of the ALU. The read operation is asynchronous with respect to all other register file controls. |
| FFEN | P5 | I | Fix-to-Float Enable. This signal controls the fixed-to-float conversion block. When FFEN is high, integer or fractional values on the DA and DB data input buses are converted to the internal floating-point format. When FFEN is low, integer, fractional, bit string, and floating-point values are input without conversion. The state of this control is latched along with the input data values. Works in conjunction with FMAT (See Table 13). |
| FMAT | R5 | I | Format Type. This signal controls the format type of the data values on input buses DA and DB. The state of this control is latched along with the input data values. Works in conjunction with FFEN (See Table 13). |
| FTBPA | D4 | I | Flow-Through Control for the Binary Point Adjust Register. When high, the BPA register is bypassed. |
| FTDAT | P13 | I | Flow-Through Control for the Input Registers on the Data Input Buses DA and DB. When high, the input registers are bypassed. |
| FTI | B16 | I | Flow-Through Control for the Multiplier and ALU Instruction Registers. When high, the instruction registers are bypassed. |
| FTMA | F17 | I | Flow-Through control for the MULTA, MULTB, ALUA, and ALUB Registers. When high, these registers will be in flow-through mode. |
| FTPS | E16 | I | Flow-Through Control for Product and Sum Output Register. When high, the product and sum registers are bypassed. |
| FTST | F14 | I | Flow-Through Control for Status Register. When high, the status register is bypassed. |

SN74ACT8867
32-BIT VECTOR PROCESSOR UNIT

Terminal Functions (Continued)

| PIN | | I/O | DESCRIPTION |
|--------|-----|-----|--|
| NAME | NO. | | |
| GND | P12 | I | Ground. All ground pins must be used and connected. |
| | G14 | | |
| | D13 | | |
| | D11 | | |
| | D10 | | |
| | D9 | | |
| | D8 | | |
| | D7 | | |
| | D6 | | |
| | D5 | | |
| | K4 | | |
| MI5 | D1 | I | Multiplier Instruction. The multiplier instruction bits are input through the instruction register in flow-through or registered mode. |
| MI6 | F2 | | |
| MI7 | C1 | | |
| MI8 | F3 | | |
| N | C16 | O | Negative Sign. The N bit is a duplicate of the sign of the result from the multiplier, ALU, or float-to-fixed block. For overflow, N indicates the direction of overflow. Also, N is used in integer and floating-point compare operations. Additional information under OES and OVER. |
| OES | D14 | I | Enable Control for the Status Output Drivers, active low. When low, the status pins will output normally. When high, they will be in an high-impedance state. |
| OEY | E14 | I | Enable Control for the Y Output Drivers, active low. When low, the Y output pins will output normally. When high, the Y outputs will be in high-impedance state. OEY also sets the PY output to high-impedance state. |
| OVER | F15 | O | Overflow. OVER is set high when an overflow condition occurs in the multiplier, ALU, or float-to-fixed block. The sign status bit indicates the direction of overflow, with N low for positive overflow and N high for negative overflow. The output data value from operations that overflow is undetermined. Additional information under OES and SELST1-SELST0. |
| PY | B4 | O | Even Parity. Even parity is generated from the final result of the float-to-fixed block and presented externally at the rising edge of the clock. Additional information under OEY. |
| RFCLK | T10 | I | Register File Clock. Data is latched for writing on the rising edge of the clock. Additional information under ENDAT, WEA, and WEB. |
| RND | C2 | I | Round Mode Select. When low, no rounding is performed in the float-to-fixed block. When high, the result of a floating-point output or conversion is rounded. RND is input through the BPA register in flow-through or registered mode. |
| SELA0 | H14 | I | Select ALU Multiplexer. Control for the ALU input multiplexers. |
| SELA1 | E17 | | |
| SELM0 | G1 | I | Select Multiplier Multiplexer. Control for the multiplier input multiplexers. |
| SELM1 | H4 | | |
| SELM2 | F1 | | |
| SELM3 | G2 | | |
| SELRFA | S9 | I | Select Register File A. Multiplexer control from write data to the A side of the register file. |
| SELRFB | P9 | I | Select Register File B. Multiplexer control from write data to the B side of the register file. |
| SELST0 | D15 | I | Select Status Source. Status from the multiplier, ALU, or float-to-fixed block may be selected for display on the status output pins by SELST1-SELST0. |
| SELST1 | D16 | | |
| SELY | C4 | I | Select Y Multiplexer. This control selects input to the output float-to-fixed block. |
| STEX | B14 | O | State of ALU/Divide by Zero. This bit indicates divide by zero from the multiplier and the state of the link flip-flop in the ALU depending on the state of the SELST1-SELST0 pins. Additional information under OES. |
| TP0 | E2 | I | Test Pin. A complete explanation is given in Table 25. |
| TP1 | B1 | | |



Terminal Functions (Continued)

| PIN | | I/O | DESCRIPTION |
|--|---|-----|---|
| NAME | NO. | | |
| VCC | K14 C11 C10 C9 C8 G4 P7 | I | Supply voltage (5 V) |
| WEA | T9 | I | Write Enable for Register File A Input, active low. When low, a write operation may be performed by asserting data and write address and toggling RFCLK. |
| WEB | R9 | I | Write Enable for Register File B Input, active low. When low, a write operation may be performed by asserting data and write address and toggling RFCLK. |
| Y0 Y1 Y2 Y3 Y4 Y5 Y6 Y7 Y8 Y9 Y10 Y11 Y12 Y13 Y14 Y15 Y16 Y17 Y18 Y19 Y20 Y21 Y22 Y23 Y24 Y25 Y26 Y27 Y28 Y29 Y30 Y31 | C15 B15 A16 C12 A15 B12 A14 B11 A13 A12 A11 A10 B10 A9 B9 B8 A8 A7 A6 A5 B7 A4 C7 A3 A2 A1 B3 B6 B2 C6 B5 C5 | O/Z | Output 32-Bit Data Bus. Output data may be converted from a floating-point to fixed-point in the float-to-fixed block. These outputs may be set to high-impedance state by the OEY pin. |
| Z | A17 | O | Zero Status. This bit is set when the result of an operation is zero. Additional information under OES. |

register file architecture

The register file of the 'ACT8867 VPU is a master-slave memory with two write ports and four read ports. The register file has 46 locations, each 40 bits wide. Each of the write and read ports are addressed by a separate 6-bit address bus. Location 0 is addressed by a zero on the address pins and subsequent locations 1 to 45 are addressed in normal binary fashion. The write address pins are labeled A5-A0 and B5-B0 and control data from the RF MUX A and RF MUX B data buses, respectively. The read addresses are labeled C5-C0, D5-D0, E5-E0, and F5-F0 and select data for the register file outputs going to the multiplier and ALU multiplexers. Read address pins C5-C0 and D5-D0 select operands for the multiplier, whereas read address pins E5-E0 and F5-F0 select operands for the ALU. Feedback to the register file from the multiplier and ALU is multiplexed into the A and B write inputs, respectively.



SN74ACT8867

32-BIT VECTOR PROCESSOR UNIT

Generally, data may be written into or read from any register file location by any input or output simultaneously. However, in the case of a simultaneous write to the same location, the data from RF MUX A is actually stored in the location, while the data from RF MUX B is ignored. Other than this, all read and write cycles may occur together in a single clock cycle. Addressing nonexistent locations is harmless, but data read from a nonexistent location will be undefined. No error status is generated from any of these conditions. The write cycle is an edge-triggered master-slave cycle with data latched on the rising edge of the register file clock (RFCLK).

Table 1. Register File Write Address Location and Input Source Selection

| WRITE ENABLE CONTROL | REGISTER FILE ADDRESS LOCATION | INPUT SOURCE |
|----------------------|--------------------------------|--------------|
| WEA low | A5-A0 | RF MUX A |
| WEB low | B5-B0 | RF MUX B |

Table 2. Register File Read Address Location and Output Destination Selection

| REGISTER FILE ADDRESS LOCATION | OUTPUT DESTINATION |
|--------------------------------|--------------------|
| C5-C0 | MULT MUX A |
| D5-D0 | MULT MUX B |
| E5-E0 | ALU MUX A |
| F5-F0 | ALU MUX B |

data inputs

Two sources, the fixed-to-float conversion block and the multiplier output, supply inputs to the RF MUX A. Input selection to RF MUX A is controlled externally by the SELRFA select line. Similarly, RF MUX B inputs come from the ALU output or the fixed-to-float conversion block. Input selection to RF MUX B is controlled externally by the SELRFB select line.

Data from either the DA or DB input bus is converted into the internal 40-bit data format of the VPU by the fixed-to-float block. Each 32-bit external input is converted into an internal format suitable to the operand type, either a floating-point number or an integer.

The fixed-to-float format conversion is selected by the FMAT control pin. The same format control is used for both inputs so only similar operand types may be input simultaneously. If FMAT is low, the input is packed as a floating-point number. If FMAT is high, it is packed as an integer for the purpose of the packing operation, signed fractional, unsigned fractional, unsigned integers, and normal integers are equivalent. Signed integers or signed fractional numbers may optionally be converted to floating-point numbers in this block, as described in fixed-to-float block section. The following examples show the data format of the signed and unsigned fractional operands:

Signed Fraction, 32 bit 2s complement.

| 31 | 30.....0 |

sign <-MSB-----LSB->

Note: magnitude of bit 30 is 2E-1.

Example: +0.125 = H'10000000'

- 0.125 = H'F0000000'

Unsigned Fraction, 32 bit.

| 31.....0 |

<-MSB-----LSB->

Note: magnitude of bit 31 is 2E-0.

Example: +0.125 = H'10000000'

+1.125 = H'90000000'

Table 3. Register File Multiplexer Input Source Selection

| RF MUX A | | RF MUX B | |
|----------|------------------------|----------|------------------------|
| SELRFA | INPUT SOURCE | SELRFB | INPUT SOURCE |
| L | product | L | sum |
| H | fixed-to-float block A | H | fixed-to-float block B |

address inputs

Each data port of the register file has a separate address control bus. Each address bus is six-bits wide, allowing a total of 64 locations to be addressed. However in the register file, there are only 48 usable locations (locations 0 through 47 decimal). The 16 addresses to nonexistent locations may be input without harmful effects, except that undefined data is output for a read operation to these invalid addresses.

The address buses A5-A0 and B5-B0 control the addresses for the two RF write inputs (see Table 1). The address inputs work in conjunction with the write enable lines (WEA and WEB) and register file clock RFCLK to perform a write operation (see Table 2). Read address buses C5-C0 and D5-D0 select input data for the A MUX and B MUX of the multiplier, respectively. Read address buses E5-E0 and F5-F0 select input data for the A MUX and B MUX of the ALU, respectively.

The RF read addresses select locations asynchronously to the write addresses, write enables, register file clock, and each other. The data stored at the addressed location will be available at the outputs after the output propagation delay, irrespective of all other control lines to the register file. Since a write changes the data stored in a location, a concurrent read from this location will depend on the write delays before the updated data will be available at the outputs. The updated data will be available after the rising edge of the RFCLK and the normal read propagation delay. A simultaneous read during a write cycle will produce the old value and a read in the subsequent cycle will produce the new value.

Locations 46 and 47 decimal of the register file are special purpose addresses used to implement the full flow-through mode of operation for the register file. Full flow-through mode is activated by setting the write enable WEA high on the A write port or WEB on the B write port and addressing location 46 decimal for A port, or location 47 decimal for B port, respectively. No memory is associated with these locations. They continually reflect the data being input on the port A and port B inputs to the register file.

Addressing location 46 decimal for any read will output the current input on port A to the selected read output. Addressing location 47 decimal for any read will output the current input on port B to the selected read output. Addressing these locations at any read port while holding WEA or WEB low will result in a partial flow-through along with an active write mode. In the partial flow-through mode, the read port reflects the condition of output of the master write latches. These master latches are transparent when RFCLK is low, and they capture the value of port A or port B on the positive edge of RFCLK. As such, the flow-through operation is disabled when RFCLK is high.

edge-triggered write mode

The write operation of the register file is implemented as an edge-triggered master-slave cycle where the state of the data inputs, write addresses, and write enables are latched on the rising edge of the register file clock (RFCLK). The new data is available to be read from the write location irrespective of the falling edge of the clock after only a propagation delay time from the rising edge. Enables for the write operations of input port A and port B are the signals WEA and WEB, respectively. These signals are active low and must be at a low logic level to perform a write operation. When the enables are at a high logic level, the associated inputs are in a full flow-through mode. No data is written to the register file while it is in full flow-through mode.

A write may be performed to any of the register file locations 0 through 45 decimal from input port A or input port B separately or concurrently. However, if a concurrent write is attempted to the same location, the data from input port A will be written and the data from input port B will be ignored. No status error is generated when data from the B port is lost during a concurrent write from the A and B ports to the same location. Addressing to locations 48 through 63 decimal performs no useful function. To perform a write operation:

1. Set up data and address, and bring write enable(s) low.
2. Toggle RFCLK (low-to-high transition).
3. Return write enable(s) to high.

If the data input register in the fixed-to-float block is enabled, then data from the input pins is latched first in this register; the data may be written to the register file on a subsequent cycle.

flow-through mode

The full flow-through mode of the register file is implemented by the use of two special purpose register file locations in which the storage latch directly connects the data lines to the register file. Therefore, they will always produce the current values of these lines when they are addressed by a read operation. Location 46 decimal is connected to the port A input and location 47 decimal is connected to the port B input. As a result, either port A or port B data is accessible to any output of the register file.

The input values for these locations depend on the state of the write enables. When a write enable is high, the input master latch for the data input is bypassed and any data applied to the port A or port B input is directly reflected in the appropriate flow-through location (see Table 4). If the write enable is low, then the value on the data lines is the data stored in the input master latch. For example, if \overline{WEA} is high and \overline{WEB} is low then register file location 46 will be the same as the data being applied to input DA (assuming FTDAT and SELRFA are high) and location 47 decimal will be the data on the input master latch of input port B. When the RFCLK is high the values of \overline{WEA} and \overline{WEB} have been latched at the rising edge and will be maintained at those values until RFCLK is low again.

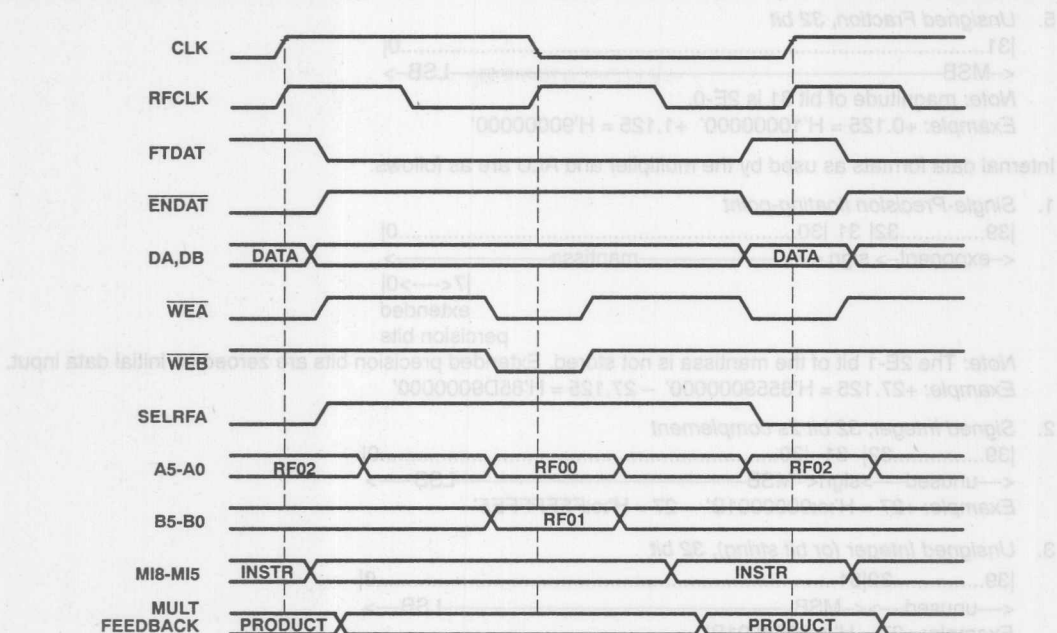
Table 4. Register File Flow-Through Mode Operations

| ADDRESS LOCATION = 46 DECIMAL | | | ADDRESS LOCATION = 47 DECIMAL | | |
|-------------------------------|-------|----------------------------------|-------------------------------|-------|----------------------------------|
| \overline{WEA} | RFCLK | DATA SOURCE | \overline{WEB} | RFCLK | DATA SOURCE |
| H | X | Port A (full flow-through) | H | X | Port B (full flow-through) |
| L | L | Port A Input Latch (transparent) | L | L | Port B Input Latch (transparent) |
| L | H | Port A Input Latch [†] | L | H | Port B Input Latch [†] |

[†] Data is latched on the rising edge of RFCLK and maintained until RFCLK is low again.

double-pumped mode

The register file is fast enough to allow operation in a double-pumped mode (see Figure 1). For each system cycle, two operands per input bus may be written to the register file by clocking the register file clock (RFCLK) at twice the speed of the main system clock. This feature is very useful in conjunction with the input fixed-to-float register. It allows a feedback write to the register file and a data input operation to be completed through each write port every cycle of the main system clock. Feedback data is written at the rising edge of the main clock and input data is latched in the fixed-to-float register at the same time. The input data is then written on the half cycle without affecting feedback operations. Using the RFCLK at twice the frequency of the main system clock also allows large amounts of data to be loaded very rapidly. However the speed of the fixed-to-float block, will not allow this double-pumping operation if the input data must be converted from integer to floating-point status.



NOTE: The above diagram shows the general relationship of control signals and timing for double-pumping the register file. Input data on DA or DB buses are latched in the data registers while feedback data from the multiplier is written into register file location RF02 at the rising edge of CLK. The latched input data is then written into register file location RF00 and RF01 at the next rising edge of RFCLK.

Figure 1. Register File Double-Pump Mode Timing Diagram

data formats

External data formats that can be input on the DA and DB pins are as follows:

1. *Single Precision floating-point, DEC F-Format*
|31.....16| 15 |14.....7|6.....0|
<-----low mantissa-----> sign <-----exponent-----> <-----high----->
mantissa
Example: +27.125 = H'000042D9' - 27.125 = H'0000C2D9'
2. *Signed Integer, 32 bit 2s complement*
| 31 | 30.....0|
sign <-----MSB----->-----LSB----->
Example: +27 = H'0000001B' - 27 = H'FFFFFFF5'
3. *Unsigned Integer (or bit string), 32 bit*
|31.....0|
<-----MSB----->-----LSB----->
Example: +27 = H'0000001B'
4. *Signed Fraction, 32 bit 2s complement*
| 31 | 30.....0|
sign <-----MSB----->-----LSB----->
Note: magnitude of bit 30 is 2E-1.
Example: +0.125 = H'10000000' - 0.125 = H'F0000000'

SN74ACT8867

32-BIT VECTOR PROCESSOR UNIT

5. Unsigned Fraction, 32 bit

|31.....0|
 <---MSB-----LSB-->

Note: magnitude of bit 31 is 2E-0.

Example: +0.125 = H'10000000' +1.125 = H'90000000'

Internal data formats as used by the multiplier and ALU are as follows:

1. Single-Precision floating-point

|39.....32| 31 |30.....0|
 <---exponent---> sign <-----mantissa----->

|7<---->0|

extended
 precision bits

Note: The 2E-1 bit of the mantissa is not stored. Extended precision bits are zeroed on initial data input.

Example: +27.125 = H'8559000000' - 27.125 = H'85D9000000'

2. Signed Integer, 32 bit 2s complement

|39.....32| 31 |30.....0|
 <---unused---> sign <---MSB-----LSB-->

Example: +27 = H'xx0000001B' - 27 = H'xxFFFFFFE5'

3. Unsigned Integer (or bit string), 32 bit

|39.....32|31.....0|
 <---unused---> <---MSB-----LSB-->

Example: +27 = H'xx0000001B'

4. Signed Fraction, 32 bit 2s complement

|39.....32| 31 |30.....0|
 <---unused---> sign <---MSB-----LSB-->

Note: magnitude of bit 30 is 2E-1.

Example: +0.125 = H'xx10000000' - 0.125 = H'xxF0000000'

5. Unsigned Fraction, 32 bit

|39.....32|31.....0|
 <---unused---> <---MSB-----LSB-->

Note: magnitude of bit 31 is 2E-0.

Example: +0.125 = H'xx10000000' +1.125 = H'xx90000000'

Data feedback from the multiplier or ALU will be stored in one of these formats. The output float-to-fixed block allows output of all of the input formats listed above plus fixed-point operands with the radix point aligned to any even bit boundary.

In general, the terms floating-point number will be used to refer to data type 1 above. Integer numbers refers to both type 2 and 3 formats. Type 4 and 5 formats are fractional numbers. Integer and fractional numbers are collectively referred to as fixed-point numbers implying a fixed radix point position. Logical operands will usually conform to the integer format. The notation F, I, and J will occasionally be used in this data sheet to represent floating-point, integer, and fractional (or fixed-point) formats, respectively.

External floating-point inputs with a zero exponent and the sign bit set are considered to be reserved operands. These reserved operands are detected and flagged accordingly in the fixed-to-float conversion blocks. Internally, floating-point numbers with a zero exponent are treated as floating-point numbers with a zero exponent and a zero mantissa. Internal floating-point numbers with a zero in the exponent, when input to the float-to-fixed block, are converted to a floating-point number with a zero in the exponent and a zero mantissa.



multiplier architecture and operation

The multiplier of the VPU is designed to perform normal multiplication on floating-point and integer operands and several other special purpose operations for Newton-Raphson division, square root, and double-precision arithmetic. The data flow diagram for the multiplier is illustrated in Figure 2.

The multiplier has two data input ports of 40 bits and one data output bus of 40 bits. In the multiplier, intermediate results are truncated rather than rounded. Error introduced by this truncation is usually insignificant for integer and fractional operations and acceptable for floating-point operations because of the extended precision. A 64-bit mantissa is retained internal to the multiplier until the last step where it is truncated to its final output form. The multiplier internally uses an extended floating-point format (40 bits) that is not DEC F-format and may therefore produce different (and possibly more accurate) answers for sequences of more than two operations done internally in floating-point format (i.e., different from the result from a nonextended format).

The VPU multiplier consists of three main blocks:

1. MULTIPLIER CORE: processes the mantissa portion of floating-point data or 32-bit signed or unsigned fixed-point operands.
2. EXPONENT ADDER/ADJUST: processes the biased exponent and adjusts it according to controls from the mantissa shift block, and then generates floating-point status.
3. MANTISSA SHIFT: normalizes the mantissa and generates the fixed-point status.

Block 1 above multiplies the low-order 32 bits of both 40-bit data input ports. Block 2 handles the high-order 8 bits when a floating-point operation is performed. The result of block 3 is to place the product output in the prescribed internal floating-point format with $1/2 \leq \text{mantissa} < 1$.

data inputs

Data for the two operands of the multiplier are selected from the following sources and by the multiplexer select codes shown in Table 5.

Table 5. Multiplier Input Operands Selection

| MULT A MUX | | | MULT B MUX | | |
|------------|-------|-----------------------|------------|-------|-----------------------|
| SELM1 | SELM0 | OPERAND SOURCE | SELM3 | SELM2 | OPERAND SOURCE |
| L | L | Multiplier output bus | L | L | Multiplier output bus |
| L | H | ALU output bus | L | H | ALU output bus |
| H | L | Seed ROM | H | L | Output numeric "1"† |
| H | H | Register file port C | H | H | Register file port D |

† The number '1' output by the B MUX will be in the correct integer or floating-point format needed as determined by the current multiplier instruction.

The DIV/SQRT pin selects the type of seed shown in Table 6.

Table 6. Seed ROM Selection

| DIV/SQRT | SEED ROM |
|----------|----------------------------|
| L | Floating-Point Division |
| H | Floating-Point Square Root |

SN74ACT8867
32-BIT VECTOR PROCESSOR UNIT

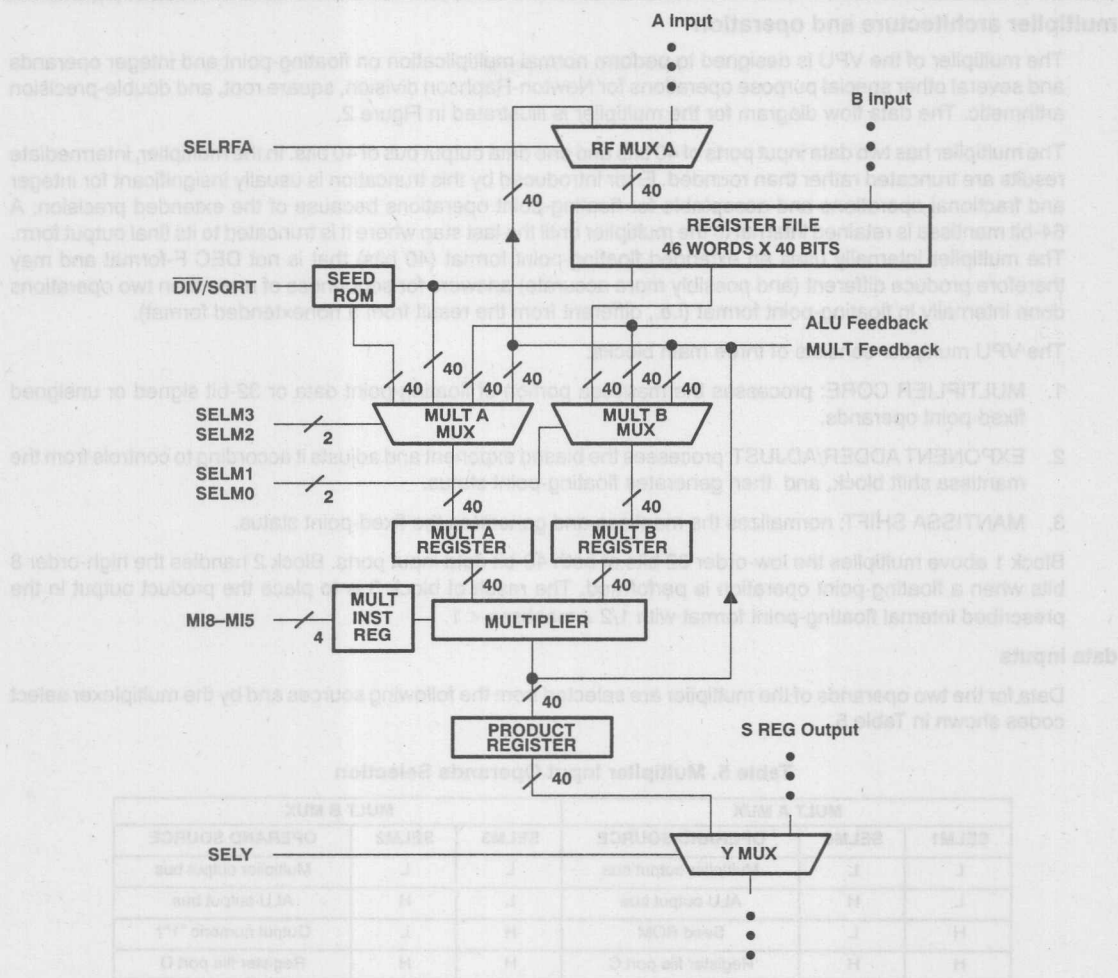


Figure 2. Multiplier Data Flow

Results from the multiplier can be routed to one or more destinations (see Table 7).

Table 7. Multiplier Output Bus Destinations

| DESTINATIONS | MULTIPLIER OUTPUT ROUTE DESCRIPTION |
|--------------------|-------------------------------------|
| Multiplier input A | MULT A MUX and MULT A register |
| Multiplier input B | MULT B MUX and MULT B register |
| ALU input A | ALU A MUX and ALU A register |
| Register file | RF MUX A† |
| Y output | Product register and Y MUX |

† Feed through of multiplier result via the register file is not allowed since direct feedback is available.

Newton-Raphson division support

The multiplier provides support for doing Newton-Raphson division using single-precision operands. Division is only supported for floating-point format due to size limitations of the seed ROM. Support consists of a seed (see Table 6) and the implementation of a special instruction to perform the necessary operations of a Newton-Raphson division cycle. This cycle performs the iterative equation:

$$X(i+1) := X(i) \times (2 - B \times X(i))$$

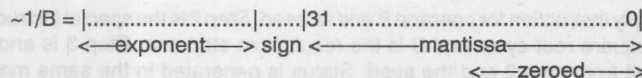
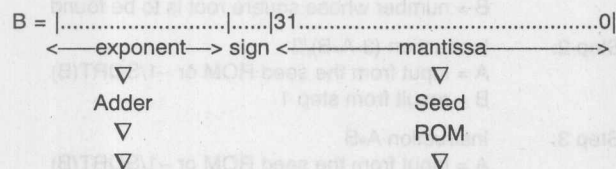
where B is the divisor and X(i) is a close approximation of the reciprocal of B. An instruction is available to perform the (2-B×X) step in one cycle without use of the ALU. The details of the implementation of this instruction and its restrictions and double-precision Newton-Raphson division is also supported and are described in Application Information section.

For floating-point operands one cycle involves two separate steps which proceeds as follows:

- Step 1: Instruction (2 - A * B)
A = input from the seed ROM or 1/B
B = divisor
- Step 2: Instruction A * B
A = input from the seed ROM or 1/B
B = result from step 1

Step one is the special instruction (2-A*B) where A is the seed for the division cycle and B is the divisor. Step two is just a normal floating-point multiplication operation. Status is generated in the same manner as with normal floating-point operations. Two cycles (four multiplies) will be required to generate a single-precision reciprocal (1/divisor) from a seed of approximately 8 bits of accuracy. The final multiply of 1/B * A (1/divisor * dividend) is also a normal floating-point multiplication operation.

The seeds for single-precision division (and square root) are generated from an internal seed ROM feeding the left multiplier input (MULT A) multiplexer that controls to the ROM select whether it is a seed for division or square root. Seeds for division have approximately eight bits of accuracy and are approximations to the actual reciprocals. Seeds for square root also have approximately eight bits of accuracy and are approximations to square root reciprocals of the numbers. The seeds are generated for optimum convergence of all input values. Consequently, the seed for certain values that can be represented exactly in the seed length will not be the exact reciprocal value since the seed is chosen to be the best for all values within the range. If desired, an external seed ROM can also be used to feed data through the DA or DB inputs and register file to the multiplier. The most significant bits of the reciprocal are critical for the Newton-Raphson process and will be set as shown below:



Seeds for square roots will be generated in the same manner except a different table for the ROM will be used (see Table 6). The exponent of a floating-point reciprocal will be produced from the biased exponent of the input using the following equations:

$$\text{Division : exponent} = 257 - \text{exponent}$$

$$\text{Square Root : exponent} = \frac{(386 - \text{exponent})}{2}$$

The biased exponent value will give legal division seed exponent values for the range 2 to 255. If the biased exponent is 0, division is an undefined operation and the multiplier will set the divide-by-zero status bit since the value on the B input will be a zero. If the biased exponent is 1, the reciprocal seed overflows and the exponent presented to the A input will be zero and the multiplier will set the divide-by-zero status bit. For square root, the allowable range for the biased exponent value is 1 to 255. An exponent of zero is an error because the Newton-Raphson square root finds the reciprocal of the square root. In this case the multiplier will also set the divide-by-zero status bit.

Newton-Raphson square root support

The multiplier provides support for doing Newton-Raphson square root; the result is actually the reciprocal of the square root for single-precision operands. Square root is only supported for floating-point format because of size limitations for the seed ROM. Support consists of a seed (see Table 6) and the implementation of a special instruction to perform the necessary operations of a Newton-Raphson square root cycle. The square root cycle performs the iterative equation:

$$X(i+1) := X(i) \times \frac{1}{2} \times (3 - B \times X(i) \times X(i))$$

where B is the number whose square root is desired and X(i) is a close approximation of the reciprocal of the square root of the number. An instruction is available to perform the $1/2 \times (3 - K \times I)$ step in one cycle without use of the ALU (K and I are the two input operands). The other instructions necessary for the square root cycle are the same as those needed for Newton-Raphson division (so no other special instructions are needed specifically for square root). Operands are in signed floating-point format but square roots of negative values are not supported. However, the sign is ignored and the resulting sign for negative values is undetermined. Double-precision Newton-Raphson square root will not be specially supported.

For floating-point operands one cycle involves three separate steps which proceeds as follows:

- Step 1: Instruction (A*B)
 A = input from the seed ROM or $\sim 1/\text{SQRT}(B)$
 B = number whose square root is to be found
- Step 2: Instruction $(3-A*B)/2$
 A = input from the seed ROM or $\sim 1/\text{SQRT}(B)$
 B = result from step 1
- Step 3: Instruction A*B
 A = input from the seed ROM or $\sim 1/\text{SQRT}(B)$
 B = result from step 2

Step 1 is a normal multiply instruction for operand B and its seed. Step 2 is the special instruction $(3-A*B)/2$ where A is the seed for the square root cycle and B is the result from step one. Step 3 is another normal multiply instruction for the result from step 2 and the seed. Status is generated in the same manner as with normal floating-point operations. A final multiply of $1/\text{SQRT}(B)*B$ to get the actual square root of a number, if needed, is also a normal floating-point multiplication operation.

Newton-Raphson division support under the multiplier section covers the description of the process using seeds (the process is the same as division). Two cycles, six multiplies, will be required to generate a single-precision reciprocal square root from a seed of approximately eight bits of accuracy.

ALU architecture and operation

The ALU of the VPU is designed to perform addition and subtraction on floating-point and integer operands, perform logic functions on bit strings, and to perform several other special operations for double-precision support. The functional diagram for the ALU is shown in Figure 3.

The ALU has two data input ports of 40 bits and an data output bus of 40 bits. The ALU internally uses an extended floating-point format (40 bits) that is not DEC F-format and may therefore produce different (and possibly more accurate) answers for sequences of more than two floating-point operations done internally. The implementation of the ALU for the VPU consists of the following main blocks:

1. EXPONENT COMPARE: compares exponents and increments the exponent of the smaller operand to equal that of the larger operand.
2. DENORMALIZE: denormalizes the smaller operand's mantissa according to the shift determined by the Exponent Compare block.
3. ALU: performs all the arithmetic and logic operations.
4. SHIFT LEFT: renormalizes the output of the ALU block.
5. EXPONENT ADJUST: adjusts the exponent due to renormalization of the mantissa.

Block 1 applies only to floating-point operands and handles the high-order 8 bits. Block 2 also applies only to floating-point operands and manipulates the mantissa section. Barrel shifting for integers is accomplished in blocks 2 and 4.

data inputs

Data for the two operands of the ALU are selected from the following sources and by the multiplexer select codes as shown in Table 8.

Table 8. ALU Input Operands Selection

| ALU A MUX | | ALU B MUX | |
|-----------|-----------------------|-----------|----------------------|
| SELA0 | OPERAND SOURCE | SELA1 | OPERAND SOURCE |
| L | Multiplier output bus | L | ALU output bus |
| H | Register file port E | H | Register file port F |

Results from the ALU can be routed to one or more of several destinations (see Table 9).

Table 9. ALU Output Bus Destinations

| DESTINATIONS | ALU OUTPUT ROUTE DESCRIPTION |
|--------------------|--------------------------------|
| Multiplier input A | MULT A MUX and MULT A register |
| Multiplier input B | MULT B MUX and MULT B register |
| ALU input A | ALU B MUX and ALU B register |
| Register file | RF MUX B† |
| Y output | Sum register and Y MUX |

† Feedthrough of ALU result via the register file is not allowed since direct feedback is available.

SN74ACT8867 32-BIT VECTOR PROCESSOR UNIT

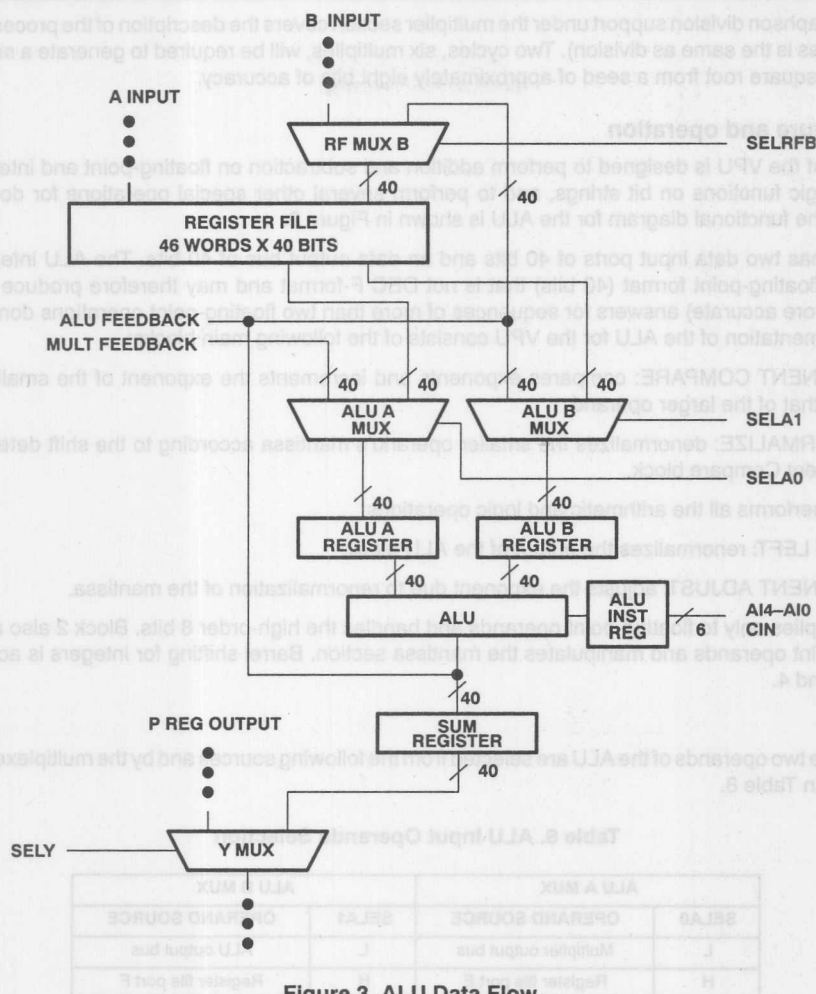


Figure 3. ALU Data Flow

data conversion operation

Data conversion operations are available in the ALU to convert data between the different formats used by the VPU (See Figure 5). However, the necessity of using the ALU to perform these functions impacts the available throughput of the vector processor and complicates microcode development. Therefore, two special data conversion blocks are located on the input and output buses of the device to perform the most commonly needed conversions and special output formatting:

- fixed-to-float conversion for inputs DA and DB.
- float-to-fixed conversion for the Y output.

Conversions may occur in parallel with multiplier and ALU operations. Note that the output of the fixed-to-float block and the input to the float-to-fixed block are in the internal data formats. The F, I, and J notations used stand for floating-point, integer, and fixed-point formats, respectively. Fixed-point formats include the format 1.31 which is also referred to as fractional.

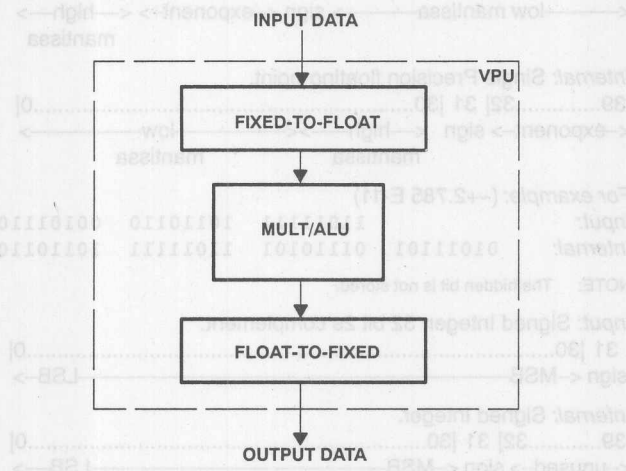


Figure 4. Conversion Blocks Data Flow

fixed-to-float block

The fixed-to-float block reformats data in the input formats to the proper internal floating-point or integer format. It also allows conversion of integers or fractional numbers to the internal floating-point format. This block has a pipeline register that may be used to pipeline data or in flow-through mode as necessary. Use of this pipeline register is required for maximum speed for input cycles where conversion from integer or fractional numbers to floating-point numbers is performed. For loading unconverted integers or floating-point numbers, this register may be left in flow-through mode. The control signals for this block are shown in Tables 10 and 11.

Table 10. Fixed-to-Float Block Data Register Operation

| FTDAT | ENDAT | DATA LATCH FUNCTION |
|-------|-------|----------------------------------|
| L | L | Latch data in register on RFCLK |
| L | H | Recirculate data in register |
| H | L | Bypass (but latch data on RFCLK) |
| H | H | Bypass data register |

Table 11. Fixed-to-Float Block Conversion Modes

| FFEN | FMAT | MNEMONIC | OPERATION |
|------|------|----------|--|
| L | L | F2F | External floating-point to internal floating-point |
| L | H | I2I | Integer to integer |
| H | L | J2F | Fractional to internal floating-point |
| H | H | I2F | Integer to internal floating-point |

When the input data registers are enabled, the control signals FFEN and FMAT are latched along with the data. When FTDAT is high and ENDAT is low, the register will latch data on the RFCLK rising edge but the flow-through data will be used internally.

SN74ACT8867

32-BIT VECTOR PROCESSOR UNIT

Integers and floating-point numbers are reformatted to the internal format of the vector processor without any loss of precision. The reformatting for floating-point and integer values is as follows:

Input: Single Precision floating-point, DEC F-Format.
 |31.....16| 15 |14.....7|6.....0|
 <-----low mantissa-----> sign <-----exponent-----> <-----high----->
 mantissa

Internal: Single Precision floating-point.
 |39.....32| 31 |30.....0|
 <-----exponent-----> sign <-----high-----> <-----low----->
 mantissa mantissa

For example: (~+2.785 E-11)

Input: 11011111 10110110 00101110 11110101
Internal: 01011101 01110101 11011111 10110110 00000000

NOTE: The hidden bit is not stored.

Input: Signed Integer, 32 bit 2s complement.
 | 31 |30.....0|
 sign <-----MSB-----> LSB-->

Internal: Signed Integer.
 |39.....32| 31 |30.....0|
 <-----unused-----> sign <-----MSB-----> LSB-->

For example: (-541184267)

Input: 11011111 10111110 00101110 11110101
Internal: 00000000 11011111 10111110 01001110 11110101

NOTE: This mode would also be used for loading fractional numbers, bit strings, unsigned integers, multiple precision, and other formats.

Conversion from integer or fractional to floating-point numbers may also be performed in this block. Because of the extended precision internal floating-point format, accuracy is not lost in this conversion. This process consists of taking the 2s complement of input values, normalizing the result, generating the exponent, and packing these pieces into the proper internal floating-point format. The following four examples are given.

Signed Integer, 32 bit 2s complement to floating-point

For example: (~-5.4 E8)

Input: 11011111 10111110 00101110 11110101
Internal: 10011110 10000001 00000111 01000100 00101110

For example: (~+1.6 E9)

Input: 01011111 10111110 00101110 11110101
Internal: 10011111 00111111 01111100 01011101 11101010

Table 11. Fixed-to-Floating Conversion Modes

| OPERATION | INTERNAL | FORMAT | FFEN |
|--|----------|--------|------|
| External floating-point to internal floating-point | FFP | L | L |
| Integer to integer | BI | H | L |
| Fractional to internal floating-point | LF | L | H |
| Integer to internal floating-point | IF | H | H |

When the input data registers are enabled, the control signals FFEN and FMT are latched along with the data. When FMT is high and ENDAT is low, the register will latch data on the FCLK rising edge but the low-through data will be used internally.

Signed Fractional Number, 32 bit 2s complement to floating-point

For example: (-0.25)

Input: 11011111 10111110 00101110 11110101
Internal: 01111111 10000001 00000111 01000100 00101100

For example: ($+0.7$)

Input: 01011111 10111110 00101110 11110101
Internal: 10000000 00111111 01111100 01011101 11101010

If a reserved floating-point operand is attempted to be input, a status bit will be set in the status register. A separate bit will show this condition for the DA and DB input buses. The reserved floating-point operand is not converted to a normal floating-point zero. This status is considered an error condition and is OR'ed into the common ERROR status bit (See Table 12).

Table 12. Fixed-to-Float Status Outputs

| SIGNAL | SELST1 | SELST0 | STATUS RESULT |
|--------|--------|--------|------------------------------|
| COUT/U | H | L | Reserved operand on DA input |
| STEX | H | L | Reserved operand on DB input |

SN74ACT8867 32-BIT VECTOR PROCESSOR UNIT

float-to-fixed block

The float-to-fixed block allows conversion from the internal floating-point format to integer and fractional formats as well as allowing output of ordinary integer and floating-point numbers. It also allows, under control of the binary point adjust pins BPA4-BPA0, alignment of the radix point (See Table 13).

Table 13. Binary Point Adjust Operation Models

| BPA4-BPA0 | MNEMONIC | FORMAT | RANGE |
|-----------|----------|---------------------|---|
| 00000 | F2J00 | 0.32 | $-0.5 \leq X < +0.5$ |
| 00001 | F2J02 | 2.30 | $-2E1 \leq X < +2E1$ |
| 00010 | F2J04 | 4.28 | $-2E3 \leq X < +2E3$ |
| 00011 | F2J06 | 6.26 | $-2E5 \leq X < +2E5$ |
| 00100 | F2J08 | 8.24 | $-2E7 \leq X < +2E7$ |
| 00101 | F2J10 | 10.22 | $-2E9 \leq X < +2E9$ |
| 00110 | F2J12 | 12.20 | $-2E11 \leq X < +2E11$ |
| 00111 | F2J14 | 14.18 | $-2E13 \leq X < +2E13$ |
| 01000 | F2J16 | 16.16 | $-2E15 \leq X < +2E15$ |
| 01001 | F2J18 | 18.14 | $-2E17 \leq X < +2E17$ |
| 01010 | F2J20 | 20.12 | $-2E19 \leq X < +2E19$ |
| 01011 | F2J22 | 22.10 | $-2E21 \leq X < +2E21$ |
| 01100 | F2J24 | 24.8 | $-2E23 \leq X < +2E23$ |
| 01101 | F2J26 | 26.6 | $-2E25 \leq X < +2E25$ |
| 01110 | F2J28 | 28.4 | $-2E27 \leq X < +2E27$ |
| 01111 | F2J30 | 30.2 | $-2E29 \leq X < +2E29$ |
| 10000 | F2I | 32.0 | $-2E31 \leq X < +2E31$ |
| 10001 | F2J | 1.31 | $-2E0 \leq X < +2E0$ |
| 10010 | F2F | Pass Floating-Point | $-2E127 \leq X < +2E127$ |
| 10011 | I2I | Pass Integer | $-2E31 \leq X < +2E31$ |
| 10100 | | Reserved | |
| . | . | . | . |
| . | . | . | . |
| . | . | . | . |
| 11101 | | Reserved | |
| 11110 | PROBEZ | PROBEZ | See Parameter Measurement Information Serial Scan Testing |
| 11111 | SCANZ | SCANZ | See Parameter Measurement Information Serial Scan Testing |

F, I, and J are notations for floating-point, integer, and fixed-point formats respectively; n.m represents 'n' bits to the left, and 'm' bits to the right of the binary point; all J formats are 2s complement numbers where the high order bit is the sign.

The radix point alignment operation may be performed only on converted floating-point values. The conversion process produces 2's complement format output from the internal floating-point format. Conversion of integers or other formats is not supported. The input to the float-to-fixed block is selected through the Y MUX according to Table 14.

Table 14. Float-to-Fixed Block Input Source Selection

| SELY | INPUT SOURCE |
|------|-------------------------------|
| L | Multiplier (Product Register) |
| H | ALU (Sum Register) |

The BPA4-BPA0 control pins and the RND pin are latched in the BPA register under control of the FTBPA and the ENBPA pins according to Table 15.

Table 15. Binary Point Adjust Register Operations

| FTBPA | ENBPA | BPA LATCH FUNCTION |
|-------|-------|-------------------------------|
| L | L | Latch BPA in register on CLK |
| L | H | Recirculate BPA in register |
| H | L | Bypass (but latch BPA on CLK) |
| H | H | Bypass BPA register |

The value on BPA4-BPA0 sets the integer field size of the fixed-point format (i.e. for 8.24 format, BPA4-BPA0 = 4 hex). The internal floating-point value is converted into this fixed-point representations. Rounding or truncation may be performed during this process. The following two examples are given for both a without rounding condition and a with rounding condition.

Without rounding:

Internal: 10000011 00110001 00000000 10110110 00111100
(floating-point)

BPA Fmt.

Output: 0.32 xxxxxxxx overflow xxxxxxxx
Output: 2.30 xxxxxxxx overflow xxxxxxxx
Output: 4.28 01011000 10000000 01011011 00011110
Output: 6.26 00010110 00100000 00010110 11000111
Output: 8.24 00000101 10001000 00000101 10110001
etc.

With rounding:

Internal: 10000011 00110001 00000000 10110110 00111100
(floating-point)

BPA Fmt.

Output: 0.32 xxxxxxxx overflow xxxxxxxx
Output: 2.30 xxxxxxxx overflow xxxxxxxx
Output: 4.28 01011000 10000000 01011011 00011110
Output: 6.26 00010110 00100000 00010110 11001000
Output: 8.24 00000101 10001000 00000101 10110010
etc.

| | | |
|------------------------|---|---|
| Post-to-fixed overflow | H | L |
| Post-to-fixed overflow | L | H |

After data and instructions are latched in the ALU or multiplier input registers, the status is available one clock cycle later including the propagation delay through the status register. The conversion process can overflow when large numbers are represented in a too small fixed-point format or from the rounding operation. For overflow, the sign bit indicates the direction of overflow. Values that are very small, i.e., zero exponent but mantissa bits not all zero, as well as reserved floating-point operands, are output as true zeros (sign bit and mantissa bits are all zero).

SN74ACT8867

32-BIT VECTOR PROCESSOR UNIT

Rounding and truncation for data conversion is completed in the float-to-fixed block. Floating-point and integer values are reformatted for output as shown below:

Internal: Single Precision floating-point.

|39.....32| 31 |30.....0|
 <--exponent--> sign <--high--> <--low-->

mantissa mantissa

Output: Single Precision floating-point, DEC F-Format.

|31.....16| 15 |14.....7|6.....0|
 <--low mantissa--> sign <--exponent--> <--high-->

mantissa

For example:

Internal: 01011101 01110101 11011111 10110110 10001100

Output: (truncated) 11011111 10110110 00101110 11110101

Output: (rounded) 11011111 10110111 00101110 11110101

NOTE: The hidden bit is not stored.

Internal: Signed Integer.

|39.....32| 31 |30.....0|
 <--unused--> sign <--MSB--> <--LSB-->

Output: Signed Integer, 32 bit 2s complement.

| 31 |30.....0|
 sign <--MSB--> <--LSB-->

For example: (no rounding is done for integers)

Internal: 00000000 11011111 10111110 00101110 11110101

Output: 11011111 10111110 00101110 11110101

NOTE: This mode would also be used for outputting fractional numbers, bit strings, unsigned integers, multiple precision, and other formats.

Status is generated for sign, zero, and overflow and is available through the status register (see Table 16).

Table 16. Float-to-Fixed Status Outputs

| SIGNAL | SELST1 | SELST0 | STATUS RESULT |
|--------|--------|--------|----------------------------|
| N | H | L | Float-to-fixed output sign |
| Z | H | L | Float-to-fixed zero result |
| OVER | H | L | Float-to-fixed overflow |

After data and instructions are latched in the ALU or multiplier input registers, the status is available one clock cycle later including the propagation delay through the status register. The conversion process can overflow when large numbers are represented in a too small fixed-point format or from the rounding operation. For overflow, the sign bit indicates the direction of overflow. Values that are very small; i.e., zero exponent but mantissa bits not all zero, as well as reserved floating-point operands, are output as true zeroes (sign bit and mantissa bits are all zero).



The status bits are generated from the following equations:

$$\begin{aligned} \text{SIGN} &= \text{B31} = 1 \\ &\quad * \text{NOT}(\text{BPA} < \text{H}'12' * \text{SHIFT} > 32 * \text{B31} = 1 * \text{NOT}(\text{EXP0})) \\ &\quad * \text{NOT}(\text{BPA} < \text{H}'12' * \text{SHIFT} = 32 * \text{B31} = 1 * \text{ROUND} = 1 * \text{M31Z}) \end{aligned}$$

The SIGN bit is the same as the sign of the internal value except for formats where a negative value is too small to be represented or rounds to zero.

$$\begin{aligned} \text{ZERO} &= (\text{BPA} = \text{H}'13' * \text{B31} = 0 * \text{M31Z}) \\ &\quad + (\text{BPA} < \text{H}'13' * \text{EXP0}) \\ &\quad + (\text{BPA} < \text{H}'12' * \text{SHIFT} > 32) \\ &\quad + (\text{BPA} < \text{H}'12' * \text{SHIFT} = 32 * \text{B31} = 0 * \text{ROUND} = 0) \\ &\quad + (\text{BPA} < \text{H}'12' * \text{SHIFT} = 32 * \text{B31} = 1 * \text{ROUND} = 1 * \text{M31Z}) \end{aligned}$$

The ZERO bit is set when an integer zero, a floating-point zero, or a reserved operand is input, or the floating-point value is too small to be represented in the selected format.

$$\begin{aligned} \text{OVER} &= (\text{BPA} = \text{H}'12' * \text{EXP1} * \text{ROUND} = 1 * \text{M24}) \\ &\quad + (\text{BPA} < \text{H}'12' * \text{SHIFT} < 0) \\ &\quad + (\text{BPA} < \text{H}'12' * \text{SHIFT} = 1 * \text{B31} = 0 * \text{ROUND} = 1 * \text{M31}) \\ &\quad + (\text{BPA} < \text{H}'12' * \text{SHIFT} = 0 * \text{B31} = 0) \\ &\quad + (\text{BPA} < \text{H}'12' * \text{SHIFT} = 0 * \text{B31} = 1 * \text{NOT}(\text{M31Z})) \end{aligned}$$

The OVER bit is set when a floating-point number is rounded and overflows or the value is too large to be represented in the selected format. Remember that the most significant bit is the sign bit for fixed-point output formats.

Where:

- BPA = value on the binary point adjust pins (4-0)
- SHIFT = $128 - \text{exponent} + (2 * \text{BPA4} - \text{BPA0})$ (for BPA < 11 hex)
or $128 - \text{exponent} + 1$ (for BPA = 11 hex)
- B31 = sign bit (bit 31)
- EXP1 = bits of the exponent (39-32) are all 1
- EXP0 = bits of the exponent (39-32) are all 0
- M31 = bits of the mantissa (30-00) are all 1
- M31Z = bits of the mantissa (30-00) are all 0
- M24 = bits of the mantissa (30-07) are all 1

rounding and truncation

If selected, the float-to-fixed block performs rounding on internal floating-point values when they are output as standard DEC F floating-point or 2s complement fixed-point format. The fixed-point output format used is selected through the BPA pins. The rounding method is standard rounding procedure with a one in the round bit rounding to the next higher value. For floating-point values output in DEC F floating-point format, this results in a round toward plus or minus infinity depending on the sign bit. The truncation is toward zero for this case. For floating-point values converted to fixed-point values, the rounding is similar whereas the truncation operation is toward minus infinity. The truncation performed in the ALU for floating-point to integer format and floating-point to fractional format is different. For these instructions, truncation is toward zero because the floating-point input is first shifted and truncated, and then 2s complemented to create the fixed-point representation. In the float-to-fixed block, the truncation is performed after the 2s complement because of implementation restrictions. Figures 5, 6, and 7 illustrate these differences along with an example.

Example: FLT(1.6) => INT(2)
FLT(1.3) => INT(1)
FLT(-.8) => INT(-1)
FLT(-.4) => INT(0)

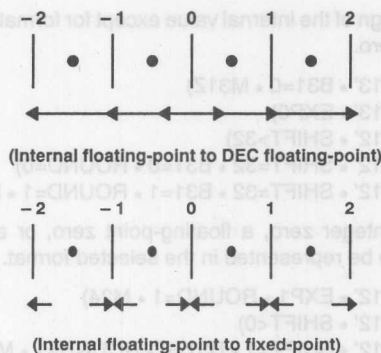


Figure 5. Float-to-Fixed Block Rounding

Example: FLT(1.6) => INT(1)
FLT(1.3) => INT(1)
FLT(-.8) => INT(-1)
FLT(-.4) => INT(-1)

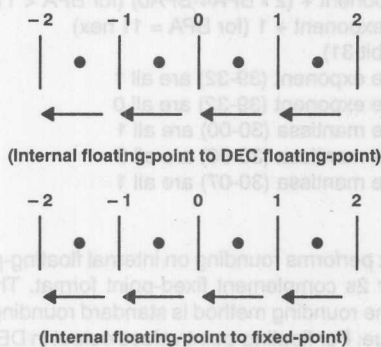


Figure 6. Float-to-Fixed Block Truncation

Example: FLT(1.6) => INT(1)
FLT(1.3) => INT(1)
FLT(-.8) => INT(0)
FLT(-.4) => INT(0)

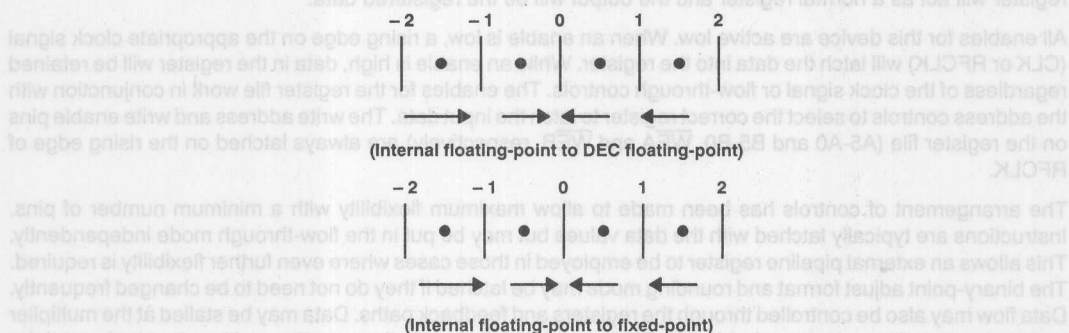


Figure 7. ALU Truncation

Rounding is performed when the RND signal is high and is not performed when the RND signal is low. The RND input is acknowledge by the BPA register. No rounding is performed when outputting integer or fractional numbers. The rounding operation may overflow and this is flagged in the appropriate bit of the status register. No other rounding is performed anywhere in the device. All multiplier and ALU results are truncated, which prevents double rounding and allows multiple-precision operations to be supported.

flow-through and pipeline mode

The data registers in the Vector Processing Unit (VPU) have been configured so that they may be enabled in pipeline mode or disabled in flow-through mode to match the supporting hardware and software configuration. Each register or closely related group of registers has its own independent controls to allow maximum flexibility. The register file uses special settings of its existing control pins to activate a flow-through type mode through this block. The entire device may be used in flow-through mode, pipelined mode, or separate sections may be independently controlled as dictated by the application requirements.

The control pins available on the VPU and the associated registers they control are itemized in Table 17.

Table 17. Enable and Flow-through Control Inputs for Registers

| REGISTER | ENABLE | FLOW-THROUGH | CLOCK |
|------------------------------------|----------|--|-------|
| Fixed-to-float A and B | ENDAT | FTDAT | RFCLK |
| Register File | WEA, WEB | WEA, WEB, and special address (46 and 47 decimal) | RFCLK |
| Mult MUX A and B | ENM | FTMA | CLK |
| Multiplier Instruction | ENM | FTI | CLK |
| ALU A and B | ENA | FTMA | CLK |
| ALU Instruction | ENA | FTI | CLK |
| Product | ENP | FTPS | CLK |
| Sum | ENS | FTPS | CLK |
| Binary Point Adjust and Round Mode | ENBPA | FTBPA | CLK |
| Status | — | FTST | CLK |

----- All flow-through controls are active high with the exception of the register file flow-through mode, which does not use a separate, explicit flow-through control pin. When a flow-through control is high, the corresponding register will be bypassed. The data on the input will appear at the output. In this mode, the enable will still control whether the bypassed register is updated on a clock pulse or not. When the flow-through control is low, the register will act as a normal register and the output will be the registered data.

All enables for this device are active low. When an enable is low, a rising edge on the appropriate clock signal (CLK or RFCLK) will latch the data into the register. While an enable is high, data in the register will be retained regardless of the clock signal or flow-through controls. The enables for the register file work in conjunction with the address controls to select the correct register to latch the input data. The write address and write enable pins on the register file (A5-A0 and B5-B0, $\overline{\text{WEA}}$ and $\overline{\text{WEB}}$, respectively) are always latched on the rising edge of RFCLK.

The arrangement of controls has been made to allow maximum flexibility with a minimum number of pins. Instructions are typically latched with the data values but may be put in the flow-through mode independently. This allows an external pipeline register to be employed in those cases where even further flexibility is required. The binary-point adjust format and rounding mode may be latched if they do not need to be changed frequently. Data flow may also be controlled through the registers and feedback paths. Data may be stalled at the multiplier or ALU inputs independently or held at the Y output while other operations continue with output to the register file. With the multiplier and ALU data latches in the flow-through mode, the computational blocks of the VPU may even be configured as a flow-through multiplier accumulator with latched or flow-through output. The status register may be selected for the registered or the feedthrough mode as desired.

status

Status for the ACT8867 is generated for error conditions, calculation results, and conversion operations. Status is latched on the rising edge of the CLK signal and will be available from the status register on the cycle subsequent to the operation where it is generated. All status bits are latched on every rising edge of the CLK signal with the exception of the Initial Error Source (IES) status bits. These bits are specially enabled by an error condition as described in Table 18. The flow-through control on the status register may be used for special applications where the status is needed as soon as it is generated. An output enable control is available to set the status output pins to high-impedance state as needed.

Both floating-point and integer operations produce status as appropriate for their operation with this status generated primarily by three main blocks: the multiplier, the ALU, and the float-to-fixed blocks. The status from each location is selected for output through the six status pins by the SELST1-SELST0 controls (see Table 18).

Table 18. Status Register Source Selection

| SELST1 | SELST0 | STATUS REGISTER SOURCE |
|--------|--------|------------------------|
| L | L | Multiplier |
| L | H | ALU |
| H | L | Float-to-fixed |
| H | H | Initial error |

The ERROR pin is set when an overflow or underflow has occurred, a divide-by-zero is attempted in the multiplier, or a reserved floating-point operand is input to the device. This status signal may not be valid for double-precision algorithms using the single-precision integer operations in the multiplier and ALU since overflows of the low-order pieces will be added to the high-order pieces in subsequent operations. When this signal becomes high, it enables the six IES status register locations to latch on the next rising edge of the CLK signal the status bits marked with a dagger (†) in Table 19, except for the NaN operand status. These status bits will then be held until the status source of the initial error is selected and released. After exiting from this state, these registers are enabled to accept status from new master errors.

Table 19. Status Register Results

| STATUS PIN | SELST1 | SELST0 | STATUS RESULT |
|------------|--------|--------|--|
| ERROR | L | L | Set when any error condition occurs in 'ACT8867 |
| | L | H | Set when any error condition occurs in 'ACT8867 |
| | H | L | Set when any error condition occurs in 'ACT8867 |
| | H | H | IES multiplier underflow |
| COUT/U | L | L | Multiplier underflow [†] |
| | L | H | ALU underflow(floating-point operation) or carry out bit(integer operation) [†] |
| | H | L | NaN on DA input [†] |
| | H | H | IES ALU underflow |
| N | L | L | Multiplier result sign |
| | L | H | ALU result sign |
| | H | L | Float-to-fixed output sign |
| | H | H | IES multiplier overflow |
| Z | L | L | Multiplier zero result |
| | L | H | ALU zero result |
| | H | L | Float-to-fixed zero result |
| | H | H | IES ALU overflow |
| OVER | L | L | Multiplier overflow [†] |
| | L | H | ALU overflow [†] |
| | H | L | Float-to-fixed overflow [†] |
| | H | H | IES float-to-fixed overflow |
| STEX | L | L | Divide by zero in multiplier [†] |
| | L | H | ALU link flip flop |
| | H | L | NaN on DB input [†] |
| | H | H | IES divide by zero |

[†] The ERROR pin is theresult of the OR of all status bits. Only ALU floating-point underflow will set this bit, not the carry-out on integer operations.

The multiplier and ALU both generate status on the result of floating-point and integer operations. The sign bit, N, will be set if the result is negative. The zero bit, Z, will be set if the result is exactly zero. Underflow will be signaled for floating-point operations when the result is too small to be represented in a normalized format. Overflow will occur for floating-point and integer results that are too large to be represented in the output format. Unsigned integer overflow for the ALU is shown by the COUT/U bit. The overflow flag will need to be ignored in microcode for special cases involving double-precision operations. The Instruction Set section has further information on multiplier operations and ALU operations.

The conversion blocks fixed-to-float and float-to-fixed generate several special status bits. The fixed-to-float blocks signal the input of a NaN value with no other status generated. The float-to-fixed block generates status on the sign and zero value of the output and overflow. The overflow bit will be set when converting a floating-point number to too small of a fixed-point format or when the rounder causes an overflow.

integer status

Integer status is generated from the multiplier and the ALU and is shown on four status pins, COUT/U, N, Z, and OVER.

Multiplier instructions operate on both 2s complement and unsigned integers and report integer status on the N, Z, and OVER signals. Underflow cannot occur for integer operations in the multiplier. In single-precision multiplication the product may be up to 64 bits in length. If it is desired to keep a 32-bit product, then any non-zero bits in bits 32 to 63 indicate the result has overflowed. Double-precision operations in the multiplier will retain all 64 bits. For example, a 32-bit by 32-bit multiplication has no overflow. The low-order multiply will appear to

overflow since the multiplier assumes a 32-bit result is wanted but this bit should be ignored in microcode at this step. The ZERO status should also be carefully interpreted since both parts must be zero before a double-precision result can be determined to be exactly zero.

The ALU reports integer status on all four of the status pins. Unsigned operations use the Z and COUT/U bits. The zero bit is set during a subtraction when both operands are equal. The COUT/U bit is set during addition indicating an overflow. The COUT/U bit goes to zero during subtraction if the result underflows. 2s complement operations use the Z, N, and OVER status bits. The Z bit is set when a 2s complement operation produces a zero result. Together the N (sign) bit and the OVER bit denote overflow. If the overflow is in the positive direction, then OVER is high and N is high. If it is negative, then OVER is high and N is low. The underflow bit is not used for 2s complement integer operations. For the multiplier, double-precision operations in the ALU can generate invalid overflow and zero status. The intermediate steps can generate valid carries from the MSB that get stored in carry flip-flops but will signal 2s complement single-precision overflow. For unsigned integers, both underflow and overflow could be incorrectly signaled. The final step of the double-precision algorithm will give a valid status. The microcode will need to handle these cases appropriately.

The float-to-fixed block generates only N and Z status bits for integer operations because there is no conversions done on integer operands and rounding cannot occur.

floating-point status

Floating-point status is generated from the multiplier, ALU, and float-to-fixed block; and is shown on the four status pins: COUT/U, N, Z, and OVER.

The ALU and multiplier report status for floating-point operations similarly. When the result of a floating-point operation is too small to be represented in a normalized format, the underflow flag (COUT/U) is asserted and the result is forced to zero. When the exponent of the result is larger than 127 decimal (unbiased), the overflow flag (OVER) is asserted and the result is undetermined. The N (sign) bit is a duplicate of the sign of the result. and has additional meaning for a compare operation.

The zero (Z) bit is set when the result is zero. However, internal to the VPU, a floating-point zero is determined by an exponent field of all zeroes, even if some mantissa bits are nonzero. During a PASS instruction, operands with zero mantissas cause zero status to be signalled, even if the exponent is nonzero. Conversely, an operand with zero exponent and nonzero mantissa is passed without signalling zero status, even though the operand will be forced to zero on output.

The float-to-fixed block converts floating-point operands to fixed-point values and generates status for this operation. The floating-point value, however, may be too big or too small to be represented in the selected fixed-point format. When the value is too large, the overflow (OVER) status bit will be set and the output value will be undefined. When a floating-point value is too small to show any nonzero bits in the fixed-point result, a zero will be output. The underflow flag is not asserted by the float-to-fixed block. Only the zero flag will be set for this situation. The rounding section of the float-to-fixed block may have a carry out of the MSB of the mantissa when outputting floating-point numbers. If so, a one bit normalization is required resulting in an increment of the exponent. If the exponent is incremented beyond 127 decimal (unbiased) an overflow will occur and OVER will be set. The SIGN and ZERO bits are set appropriately for floating-point format and converted fixed-point output.

integer compare

The ALU instruction set does not include an integer compare instruction. However, the ALU status reported for the integer subtraction instruction ($A + \bar{B} + \text{CIN}$), with CIN pin being set high provides the information necessary to compare the magnitude of two unsigned or 2s complement numbers. This instruction will set the N (sign), zero (Z), and overflow (OVER) status bits. The interpretation of the status bits required to determine the relation of the two operands is given in Table 20.

Table 20. Integer Compare Status Results

| RELATION | UNSIGNED NUMBERS | | 2s COMPLEMENT NUMBERS | |
|----------|--------------------|-------|---------------------------|-------|
| | STATUS BITS | STATE | STATUS | STATE |
| A EQ B | Z | HIGH | Z | HIGH |
| A NE B | Z | LOW | Z | LOW |
| A GE B | COUT/U | HIGH | N XNOR OVER | HIGH |
| A LT B | COUT/U | LOW | N XOR OVER | HIGH |
| A GT B | COUT/U AND (NOT Z) | HIGH | (N XNOR OVER) AND (NOT Z) | HIGH |
| A LE B | (NOT COUT/U) OR Z | HIGH | (N XOR OVER) OR Z | HIGH |

floating-point compare

The status from the compare A and B instruction (CMP) is generated by the ALU to provide a method for determining the relationship of two floating-point values. The normal floating-point subtract operation, because of underflow and overflow, cannot generate complete status for all cases. The compare operation will set the status correctly for the values of the operands. The underflow and overflow bits are never set by this instruction since the operation being performed is a compare and not a true subtraction. The instruction generates the N (sign) and Z (zero) bits, which are interpreted in a similar manner to the integer compare above to determine the relationship of the numbers. No meaningful data is output when this operation is executed.

The algorithm used by the compare instruction first compares the signs of the two operands. If the signs are the same, then a subtraction will be performed. Otherwise the status bits can be determined directly. Table 21 shows the complete status generation algorithm.

| | | |
|--------|---------------------------|------|
| A EQ B | Z | HIGH |
| A NE B | Z | LOW |
| A GE B | N XNOR OVER | HIGH |
| A LT B | N XOR OVER | HIGH |
| A GT B | (N XNOR OVER) AND (NOT Z) | HIGH |
| A LE B | (N XOR OVER) OR Z | HIGH |

Table 21 shows the complete status generation algorithm. The status bits can be determined directly. Table 21 shows the complete status generation algorithm.

Table 22. Test Pin Control Inputs

| TEST | TPS | OPERATION |
|------|-----|-------------------------------|
| L | L | Outputs driven low |
| L | H | Outputs driven high |
| H | L | Outputs driven high-impedance |
| H | H | Normal device operation |

Table 21. Floating-Point Compare Status Generation Algorithm

| SIGN A | SIGN B | OPERATION | ALGORITHM |
|--------|--------|----------------|---|
| - | - | $ A - B $ | OVER is low IF result < 0 THEN N is high (A LT B) ELSE N is low (A GE B) IF result = 0 THEN Z is high (A EQ B) ELSE Z is low (A NE B) |
| + | - | $ A + B $ | OVER is low; N is low; Z is low; result is always > 0 (A GT B) |
| + | + | $ A - B $ | OVER is low IF result < 0 THEN N is high (A LT B) ELSE N is low (A GE B) IF result = 0 THEN Z is high (A EQ B) ELSE Z is low (A NE B) |
| - | + | $- A \pm B $ | OVER is low; N is high; Z is low; result is always < 0 (A LT B) |

The interpretation of the status for the floating-point compare instruction is given in Table 22.

Table 22. Floating-Point Compare Status Results

| RELATION | FLOATING-POINT NUMBER | |
|----------|---------------------------|-------|
| | STATUS [†] | STATE |
| A EQ B | Z | HIGH |
| A NE B | Z | LOW |
| A GE B | N XNOR OVER | HIGH |
| A LT B | N XOR OVER | HIGH |
| A GT B | (N XNOR OVER) AND (NOT Z) | HIGH |
| A LE B | (N XOR OVER) OR Z | HIGH |

[†] OVER (and COUT/U) will always be low. It has been included in the above equations to provide compatibility between these and the 2s complement relationships.

data and system integrity

output parity

Even parity is generated from the final result of the float-to-fixed block and presented externally at the output PY. When output enable \overline{OEY} is high, this output is in the high-impedance state. When \overline{OEY} is low, it is in the normal output mode.

test pins

Two test pins on the VPU control the device test modes and are given in Table 23.

Table 23. Test Pin Control Inputs

| TP1 | TP0 | OPERATION |
|-----|-----|-------------------------------|
| L | L | Outputs driven low |
| L | H | Outputs driven high |
| H | L | Outputs driven high-impedance |
| H | H | Normal device operation |

The primary purpose of the test pins is for testability requirements by Texas Instruments. They are useful for fault-tolerant systems to allow a malfunctioning part to be effectively removed from the system by forcing all its outputs to a high-impedance state.

absolute maximum ratings over operating free-air temperature range (unless otherwise noted)[†]

| | |
|--|-----------------|
| Supply voltage, V_{CC} (see Note 1) | – 0.5 V to 7 V |
| Input clamp current, I_{IK} ($V_I < 0$ or $V_I > V_{CC}$) | ± 20 mA |
| Output clamp current, I_{OK} ($V_O < 0$ or $V_O > V_{CC}$) | ± 50 mA |
| Continuous output current, I_O ($V_O = V_{CC}$) | ± 50 mA |
| Continuous current through V_{CC} or GND pins | ± 100 mA |
| Operating free-air temperature range | 0°C to 70°C |
| Storage temperature range | – 65°C to 150°C |

[†] Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTE 1: All voltage levels are with respect to GND.

recommended operating conditions

| | | MIN | NOM | MAX | UNIT |
|----------|--------------------------------|-------|-----|----------|------|
| V_{CC} | Supply voltage | 4.75 | 5 | 5.25 | V |
| V_{IH} | High-level input voltage | 2 | | V_{CC} | V |
| V_{IL} | Low-level input voltage | – 0.3 | | 0.8 | V |
| I_{OH} | High-level output current | | | – 8 | mA |
| I_{OL} | Low-level output current | | | 8 | mA |
| T_A | Operating free-air temperature | 0 | | 70 | °C |

electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

| PARAMETER | TEST CONDITIONS | MIN | TYP | MAX | UNIT |
|--|---|--|-----|------|------|
| V_{OH} High-level output voltage | $V_{CC} = 5.25$ V, $I_{OH} = -8$ mA | 4.26 | | | V |
| | $V_{CC} = 4.75$ V, $I_{OH} = -8$ mA | 3.76 | | | |
| V_{OL} Low-level output voltage | $V_{CC} = 5.25$ V, $I_{OL} = 8$ mA | | | 0.45 | V |
| | $V_{CC} = 4.75$ V, $I_{OL} = 8$ mA | | | 0.45 | |
| I_I Input current | All Others | $V_{CC} = 5.25$ V, $V_I = V_{CC}$ to 0 | | ± 5 | µA |
| | TP0 & TP1 | $V_{CC} = 5.25$ V, $V_I = V_{CC}$ to 0 | | ± 25 | |
| I_Z I/O or output off-state current | $V_{CC} = 5.25$ V, $V_I = V_{CC}$ to 0, I/O or output off | | | ± 10 | µA |
| I_{CCQ} Quiescent supply current | $V_{CC} = 5.25$ V, $V_I = 0.2$ V to $V_{CC} - 0.2$ V | | | 1 | mA |
| I_{CC}^{\ddagger} Operating supply current | $V_{CC} = 5.25$ V, $V_I = 0.2$ V to $V_{CC} - 0.2$ V, $I_O = 0$ | | | 200 | mA |
| C_i Input capacitance | | | 10 | | pF |

[‡] I_{CC} is measured with maximum clock frequency equivalent to 16 MHz. Inputs should be presented with random logic highs and lows to assure the toggling of internal nodes. Outputs are presented with capacitive loads of 50 pF.

SN74ACT8867 32-BIT VECTOR PROCESSOR UNIT

switching characteristics over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

| | PARAMETER | MIN | TYP† | MAX | UNIT |
|-------------------|---|-----|------|-----|------|
| t _{p1} | Propagation delay, clock to product/sum registers to Y outputs, all BPA cases | | 30 | 45 | ns |
| t _{p2} | Propagation delay, clock to MULT/ALU registers to status outputs with FTST high | | 42 | 58 | ns |
| t _{p3} | Propagation delay, clock to status registers to status outputs with FTST low | | 16 | 24 | ns |
| t _{p4} | Propagation delay, clock to MULT/ALU registers and flow-through to Y bus with FTMA = ENM = ENA = low and FTPS = FTBPA = high | | 44 | 70 | ns |
| t _{p5} | Propagation delay, clock to register file and flow-through to Y bus (read same location written) with WEA = WEB = low and FTMA = FTPS = FTBPA = high | | 50 | 80 | ns |
| t _{p6} | Propagation delay, clock to DA/DB registers and flow-through to Y bus, with WEA = WEB = high, RA = 2E hex, RB = 2F hex, FTMA = FTPS = FTBPA = high, and fixed-to-float inactive | | 50 | 80 | ns |
| t _{p7} | Propagation delay, clock to DA/DB registers and flow-through to Y bus, with WEA = WEB = high, RA = 2E hex, RB = 2F hex, FTMA = FTPS = FTBPA = high, and fixed-to-float active | | 60 | 90 | ns |
| t _{p8} | Propagation delay, DA/DB inputs to Y bus, with WEA = WEB = high, RA = 2E hex, RB = 2F hex, and FTDAT = FTMA = FTPS = FTBPA = high | | 55 | 85 | ns |
| t _{p9} | Propagation delay, SELY to Y bus, float-to-fixed in pass mode | | 20 | 30 | ns |
| t _{p10} | Propagation delay, CLK and SELY to Y bus, all BPA cases, with FTPS = ENP = ENS = low | | 25 | 42 | ns |
| t _{p11} | Propagation delay, SELY to Y bus, all BPA cases including float-to-fixed conversion with binary point adjust, with FTPS = ENP = ENS = low | | 28 | 37 | ns |
| t _{p12} | Propagation delay, BPA to Y bus, all BPA cases with FTBPA high | | 30 | 45 | ns |
| t _{p13} | Propagation delay, MI and AI instruction bits to Y bus with FTPS high | | 30 | 62 | ns |
| t _{p14} | Propagation delay, read address ports C, D, E, or F to Y bus, with FTMA = FTPS = high | | 48 | 75 | ns |
| t _{p15} | Propagation delay, read address ports C, D, E, or F to Y bus through seed ROM, with FTMA = FTPS = high | | 55 | 85 | ns |
| t _{p16} | Propagation delay, SELST to status output | | 15 | 25 | ns |
| t _{en1} | Enable time, OEY to Y data outputs | | 12 | 20 | ns |
| t _{en2} | Enable time, OES to status outputs | | 12 | 20 | ns |
| t _{dis1} | Disable time, OEY to Y data outputs | | 12 | 20 | ns |
| t _{dis2} | Disable time, OES to status outputs | | 12 | 20 | ns |

† All typical values are characterized at 5 V and 25°C.

| | | | |
|----|-----|--|---------------------------------|
| Am | 1 | VCC = 5.5 V, V _{CC} = 5.0 V, V _{CC} = 4.5 V, V _{CC} = 4.0 V, V _{CC} = 3.5 V, V _{CC} = 3.0 V, V _{CC} = 2.5 V, V _{CC} = 2.0 V, V _{CC} = 1.5 V, V _{CC} = 1.0 V, V _{CC} = 0.5 V, V _{CC} = 0.0 V | I/O or output or status current |
| Am | 500 | VCC = 5.5 V, VCC = 5.0 V, VCC = 4.5 V, VCC = 4.0 V, VCC = 3.5 V, VCC = 3.0 V, VCC = 2.5 V, VCC = 2.0 V, VCC = 1.5 V, VCC = 1.0 V, VCC = 0.5 V, VCC = 0.0 V | Quiescent supply current |
| Am | 500 | VCC = 5.5 V, VCC = 5.0 V, VCC = 4.5 V, VCC = 4.0 V, VCC = 3.5 V, VCC = 3.0 V, VCC = 2.5 V, VCC = 2.0 V, VCC = 1.5 V, VCC = 1.0 V, VCC = 0.5 V, VCC = 0.0 V | Operating supply current |
| Am | 10 | VCC = 5.5 V, VCC = 5.0 V, VCC = 4.5 V, VCC = 4.0 V, VCC = 3.5 V, VCC = 3.0 V, VCC = 2.5 V, VCC = 2.0 V, VCC = 1.5 V, VCC = 1.0 V, VCC = 0.5 V, VCC = 0.0 V | Input capacitance |

Am is measured with maximum clock frequency equivalent to 10 MHz. Inputs should be presented with random high and low states. Outputs are presented with capacitive load. Am 500 to check for excessive loading.

timing requirements over recommended ranges of supply voltage and operating free-air temperature

| | PARAMETER | MIN | MAX | UNIT |
|-------------------|--|-----|-----|------|
| t _{c1} | Cycle time, CLK [†] | 62 | | ns |
| t _{c2} | Cycle time, RFCLK [‡] | 31 | | ns |
| t _{w1} | Pulse duration, CLK high | 10 | | ns |
| t _{w2} | Pulse duration, CLK low | 12 | | ns |
| t _{w3} | Pulse duration, RFCLK high | 10 | | ns |
| t _{w4} | Pulse duration, RFCLK low | 12 | | ns |
| t _{su1} | Setup time, DA/DB data bus input before RFCLK | 10 | | ns |
| t _{su2} | Setup time, register file write address A/B port before RFCLK | 12 | | ns |
| t _{su3} | Setup time, register file enables WEA/WEB before RFCLK | 10 | | ns |
| t _{su4} | Setup time, DA/DB register enables before RFCLK | 10 | | ns |
| t _{su5} | Setup time, register enables (ENM, ENP, ENS, and ENA) before CLK | 10 | | ns |
| t _{su6} | Setup time, SELM, SELA before CLK with ENM = ENA = low | 10 | | ns |
| t _{su7} | Setup time, instruction bits MI/AI before CLK with ENM = ENA = low | 10 | | ns |
| t _{su8} | Setup time, binary point adjust BPA before CLK with ENBPA low | 10 | | ns |
| t _{su9} | Setup time, select status SELST before CLK | 12 | | ns |
| t _{su10} | Setup time, round input RND before CLK with ENBPA low | 10 | | ns |
| t _{su11} | Setup time, carry input CIN before CLK with ENA low | 10 | | ns |
| t _{su12} | Setup time, select register file MUX SELRFA/B before RFCLK | 10 | | ns |
| t _{su13} | Setup time, fixed-to-float enable FFEN before RFCLK | 10 | | ns |
| t _{su14} | Setup time, fixed-to-float format FMAT before RFCLK | 10 | | ns |
| t _{h1} | Hold time, DA/DB data bus input after RFCLK | 6 | | ns |
| t _{h2} | Hold time, register file write address A/B port after RFCLK | 2 | | ns |
| t _{h3} | Hold time, register file enables WEA/WEB after RFCLK | 2 | | ns |
| t _{h4} | Hold time, DA/DB register enables after RFCLK | 2 | | ns |
| t _{h5} | Hold time, register enables (ENM, ENP, ENS, and ENA) after CLK | 2 | | ns |
| t _{h6} | Hold time, SELM, SELA after CLK with ENM = ENA = low | 4 | | ns |
| t _{h7} | Hold time, instruction bits MI/AI after CLK with ENM = ENA = low | 6 | | ns |
| t _{h8} | Hold time, binary point adjust BPA after CLK with ENBPA low | 6 | | ns |
| t _{h9} | Hold time, select status SELST after CLK | 6 | | ns |
| t _{h10} | Hold time, round input RND after CLK with ENBPA low | 6 | | ns |
| t _{h11} | Hold time, carry input CIN after CLK with ENA low | 6 | | ns |
| t _{h12} | Hold time, select register file MUX SELRFA/B after RFCLK | 2 | | ns |
| t _{h13} | Hold time, fixed-to-float enable FFEN after RFCLK | 5 | | ns |
| t _{h14} | Hold time, fixed-to-float format FMAT after RFCLK | 5 | | ns |

[†] Fully pipelined, rising edge to rising edge

[‡] Permits two write cycles to register file during one CLK period (double-pumping)

TYPICAL CHARACTERISTICS

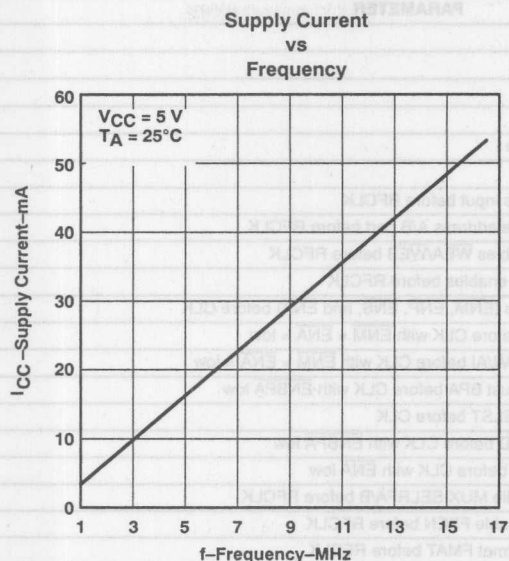


Figure 8

PARAMETER MEASUREMENT INFORMATION

serial scan testing mechanism

Texas Instruments provides the scan-testing hardware features described here. The hardware for scan testing included on the 'ACT8867 is designed to reduce the enormous problem of fault detection for the whole device to several more manageable fault detection tasks. These tasks are divided among the register file, the multiplier core, the ALU core, the fixed-to-float logic, and the float-to-fixed logic as isolatable sections.

Features of Scan Test Mechanism:

1. Capability to serially load and serially dump all registers except the instruction register, status register, and the BPA register for development and improvement of patterns to a high grade level. Because of the prohibitive hardware complexity that would result, the register file is also excluded from scanning.
2. Probe testing with contact required to less than 100 of the 208 device pins.
3. Three serially loadable registers to drive input buses: DAFMT, DBFMT, RFADDR
4. One serial output register to read Y Bus: YSCAN
5. Minimal interaction between critical delay path circuits and test circuitry
6. Ten total serial input and output scan paths with a maximum of 40 bits each

The serial scan chains are implemented as illustrated in Table 24.



PARAMETER MEASUREMENT INFORMATION

Table 24. Serial Scan Chains

| REGISTER | SERIAL INPUT | SERIAL OUTPUT | NUMBER OF FFs |
|----------|---------------------------|---------------|---------------|
| MULT A | SELM0 | ERROR | 40 |
| MULT B | SELM1 | COUT/U | 40 |
| ALU A | SELA0 | N | 40 |
| ALU B | SELA1 | Z | 40 |
| PRODUCT | SELM3 | OVER | 40 |
| SUM | CIN | STEX | 40 |
| RFADDR | A5 | Y3 | 36 |
| DAFMT | DA31 | Y2 | 32 |
| DBFMT | DB31 | Y1 | 32 |
| YSCAN | Serial input is logic '0' | Y0 | 32 |

serial scan testing control signals

Two internal control signals will be generated to initiate scan testing. The first signal, SCANZ, will be decoded from the state BPA4-BPA0 is 1F hex and TP1-TP0 is 0 hex. When SCANZ is active (low), all ten scan registers listed in Table 24 will be in a serial-in, serial-out mode. They may be simultaneously read out and reloaded with their expected contents by a series of 40 clocks. The serial input and serial output pin assignments are given in Table 24.

The second signal, PROBEZ, will be decoded from the state BPA4-BPA0 is 1E hex and TP1-TP0 is 0 hex. When PROBEZ is active (low), seven of the ten registers listed in Table 24 are in a normal operating mode. The normal mode means that the registers are fully under control of their respective feed-through and enable pins. The Y-SCAN register simply samples the data being routed to the pins Y31-Y0.

Only DAFMT, DBFMT, and RFADDR are placed in a hold mode and the inputs to the device normally coming from DA31-DA0 and DB31-DB0 are supplied instead by the corresponding bits stored in registers DAFMT and DBFMT respectively. The register file addresses normally supplied by way of pins WA5-WA0, WB5-WB0, RC5-RC0, RD5-RD0, RE5-RE0, and RF5-RF0 are instead supplied by the corresponding bits stored in the RFADDR register.

The PROBEZ control coupled with the SCANZ control allows for sequences of tests using only one pin for DA31-DA0, one pin for DB31-DB0, and one pin for the register file addresses. Another pin would be used for the output bus Y31-Y0. This potentially reduces the probe pin count by 124 pins or to a total of 84 pins and provides access to the contents of all the major registers on the device as listed in Table 24. The sequence would be as follows:

1. SCANZ active: load and read all registers using 40 clock sequence.
2. PROBEZ active: clock once to advance data one pipeline position.
3. SCANZ active: load and read all register using 40 clock sequence.

PARAMETER MEASUREMENT INFORMATION

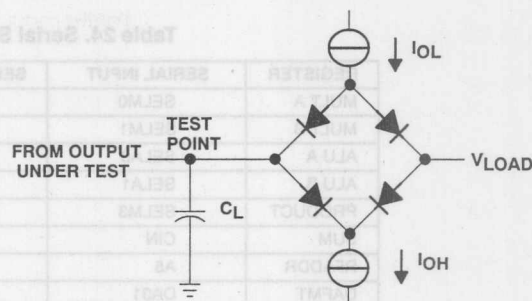
LOAD CIRCUIT PARAMETERS

| TIMING PARAMETERS | | C _{LOAD} [†] (pF) | I _{OL} (mA) | I _{OH} (mA) | V _{LOAD} (V) |
|----------------------|------------------|--|-------------------------|-------------------------|--------------------------|
| t _{en} | t _{PZH} | 50 | 8 | -8 | 0 |
| | t _{PZL} | | | | 3 |
| t _{dis} | t _{PHZ} | 50 | 8 | -8 | 1.5 |
| | t _{PLZ} | | | | |
| t _{pd} | | 50 | 8 | -8 | ± |

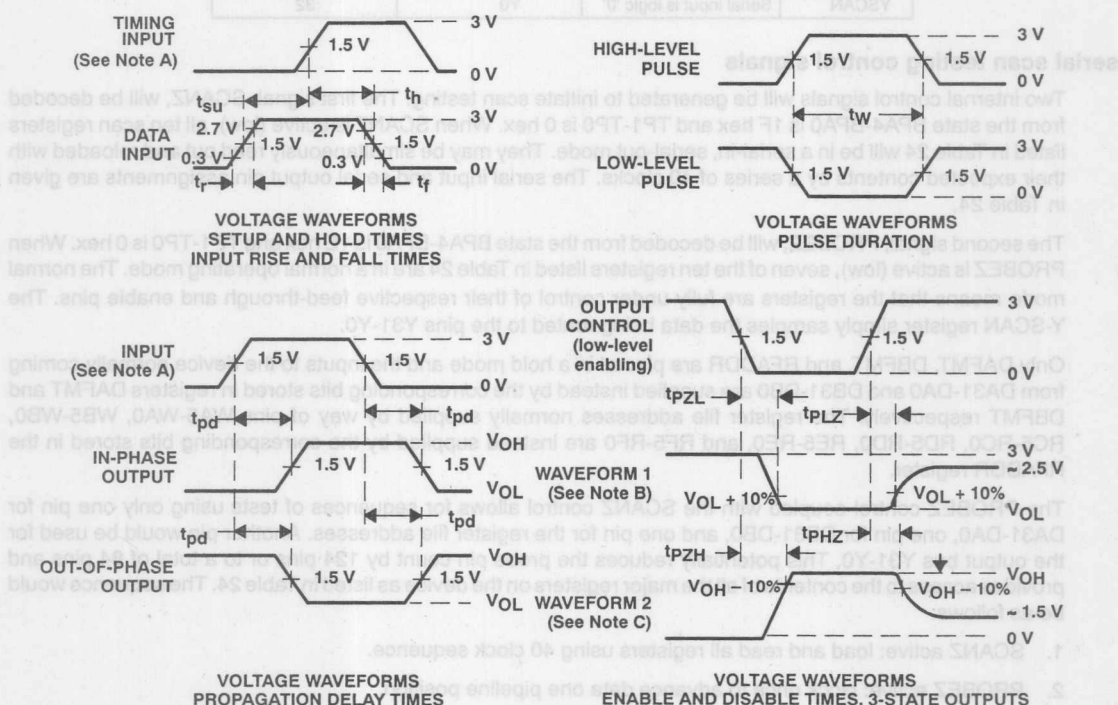
† $C_{I,OAD}$ includes probes and test fixture capacitance.

 $\pm V_{I,OAD} - V_{OI} = 50 \Omega$, where $V_{OI} = 0.45 \text{ V}$, $I_{OI} = 8 \text{ mA}$.

101



LOAD CIRCUIT



NOTES: A. Phase relationships between waveforms were chosen arbitrarily. All input pulses are supplied by pulse generators having the following characteristics: PRR = 1 MHz, $Z_0 = 50 \Omega$, $t_r \leq 6$ ns, $t_f \leq 6$ ns.

B. Waveform 1 is for an output with internal conditions such that the output is low except when disabled by the output control.

C. Waveform 2 is for an output with internal conditions such that the output is high except when disabled by the output control. For t_{PLZ} and t_{PHZ} , V_{OL} and V_{OH} are measured values.

Figure 9

PROGRAMMING INFORMATION

multiplier instruction set

Multiplier operations are under control of instruction bits M18-M15. Bit 8 signifies integer or floating-point operations as follows:

Table 25. Multiplier Instruction Set

| INSTRUCTION BITS | | MULTIPLIER |
|------------------|---------|------------------|
| M18 | M17-M15 | OPERATIONS |
| 0† | 000 | $+(A * B)$ |
| | 001 | $-(A * B)$ |
| | 010 | $+(A * B)$ |
| | 011 | $-(A * B)$ |
| | 100 | $+(A * B)$ |
| | 101 | $-(A * B)$ |
| | 110 | $(2 - A * B)$ |
| | 111 | $(3 - A * B)/2$ |
| 1‡ | 000 | LS $A * B$ |
| | 001 | MS $A * B$ |
| | 010 | LS $A * \#B$ |
| | 011 | MS $A * \#B$ |
| | 100 | LS $\#A * \#B$ |
| | 101 | MS $\#A * \#B$ |
| | 110 | LS $M(A) * M(B)$ |
| | 111 | MS $M(A) * M(B)$ |

† Floating-Point Format

‡ Integer Format

NOTE: A pass A operation exists for the multiplier by forcing a '1' to be input on the B input port by selecting SELM(3-2) = B'10'. The notation "#A" for integer operations means the input value is assumed to be unsigned. The notation "M(A)" means the input value is assumed to be unsigned but the MSB (floating-point hidden bit) is forced to a 1.

The following text gives a more complete description of each of the multiplier opcodes and their operation. The general format used is:

STATUS
N: Sign
Z: Zero
COUT/U: Underflow
OVER: Overflow
STEX: Divide-by-Zero

INSTRUCTION: $+(A * B)$ OPCODE: MI = 0 hex

DESCRIPTION: Multiply floating-point. The lower 32 bits of the mantissa are truncated after normalization. The sign of the product is the exclusive-OR combination of the sign of the operands.

MNEMONIC: MF

STATUS

N: The sign of the product
Z: Zero flag is set when either A or B is zero (i.e. the exponent = 0 and the sign = 0). Sign and exponent of the product are set to zero.
COUT/U: Underflow flag is set when the exponent of the product is zero or negative. Sign and exponent of the product are set to zero.
OVER: Overflow flag is set when the exponent of the product is larger than FF hex.

INSTRUCTION: $-(A * B)$ OPCODE: MI = 1 hex

DESCRIPTION: Multiply floating-point Negate output. The lower 32 bit of the mantissa is truncated after normalization. The sign of the product is the exclusive-NOR combination of the sign of the operands.

MNEMONIC: MFN

STATUS

N: The sign of the product.
Z: Zero flag is set when either A or B is zero (i.e. the exponent = 0 and the sign = 0). Sign and exponent of the product are set to zero.
COUT/U: Underflow flag is set when the exponent of the product is zero or negative. Sign and exponent of the product are set to zero.
OVER: Overflow flag is set when the exponent of the product is larger than FF hex.

INSTRUCTION: $+(A * |B|)$ OPCODE: MI = 2 hex

DESCRIPTION: Multiply floating-point using absolute value of B. The lower 32 bit of the mantissa is truncated after normalization. The sign of the product is that of the input in A port.

MNEMONIC: MFAB

STATUS

N: The sign of the product.
Z: Zero flag is set when either A or B is zero (i.e. the exponent = 0 and the sign = 0). Sign and exponent of the product are set to zero.
COUT/U: Underflow flag is set when the exponent of the product is zero or negative. Sign and exponent of the product are set to zero.
OVER: Overflow flag is set when the exponent of the product is larger than FF hex.

PROGRAMMING INFORMATION

INSTRUCTION: $-(A * |B|)$ OPCODE : MI = 3 hex

DESCRIPTION : Multiply floating-point using absolute value of B and Negate output. The lower 32 bit of the mantissa is truncated after normalization. The sign of the product is the inversion of that of the input in A port.

MNEMONIC: MFABN

STATUS

N: The sign of the product.

Z: Zero flag is set when either A or B is zero (i.e. the exponent = 0 and the sign = 0). Sign and exponent of the product are set to zero.

COUT/U: Underflow flag is set when the exponent of the product is zero or negative. Sign and exponent of the product are set to zero.

OVER: Overflow flag is set when the exponent of the product is larger than FF hex.

INSTRUCTION: $+(|A| * |B|)$ OPCODE : MI = 4 hex

DESCRIPTION : Multiply floating-point using absolute value of A and B. The lower 32 bit of the mantissa is truncated after normalization. The sign of the product is 0.

MNEMONIC: MFABS

STATUS

N: The sign of the product.

Z: Zero flag is set when either A or B is zero (i.e. the exponent = 0 and the sign = 0). Sign and exponent of the product are set to zero.

COUT/U: Underflow flag is set when the exponent of the product is zero or negative. Sign and exponent of the product are set to zero.

OVER: Overflow flag is set when the exponent of the product is larger than FF hex.

INSTRUCTION: $-(|A| * |B|)$ OPCODE : MI = 5 hex

DESCRIPTION : Multiply floating-point using absolute value of A and B Negate output. The lower 32 bit of the mantissa is truncated after normalization. The sign of the product is 1.

MNEMONIC: MFABSN

STATUS

N: The sign of the product.

Z: Zero flag is set when either A or B is zero (i.e. the exponent = 0 and the sign = 0). Sign and exponent of the product are set to zero.

COUT/U: Underflow flag is set when the exponent of the product is zero or negative. Sign and exponent of the product are set to zero.

OVER: Overflow flag is set when the exponent of the product is larger than FF hex.

PROGRAMMING INFORMATION

INSTRUCTION: (2 - A * B) OPCODE : MI = 6 hex

DESCRIPTION : Two Minus A Times B. The output is equal to 2 minus the product of A and B with the lower 32 bit of the mantissa truncated after normalization. This is the special instruction used in Newton-Raphson approximation of 1/B. The product A * B must be approximately 1 for this instruction to execute properly since the normalization circuit for the subtract operation is based on this assumption and cannot normalize the result for unrestricted values of A and B.

MNEMONIC: NRDIV

STATUS

N: The sign of the product.

Z: Zero flag is set when either A or B is zero (i.e. the exponent = 0 and the sign = 0). Sign and exponent of the product are set to zero.

COUT/U: Underflow flag is set when the exponent of the product is zero or negative. Sign and exponent of the product are set to zero.

OVER: Overflow flag is set when the exponent of the product is larger than FF hex.

STEX: Divide by Zero. This status bit is set when A or B are zero (i.e. exponent is zero). This will be the case when the exponent of B is 0 hex or 1 hex. The first case is a divide by zero and the second causes an intermediate overflow for which the seed rom sends out a zero exponent on A, which sets this flag.

INSTRUCTION: (3 - A * B)/2 OPCODE : MI = 7 hex

DESCRIPTION : One Half of Three Minus A Times B. The output is equal one half of 3 minus the product of A and B with the lower 32 bit of the mantissa truncated after normalization. This is the special instruction used in Newton-Raphson approximation of 1/Square Root of B. The product A * B must be approximately 1 for this instruction to execute properly since the normalization circuit for the subtract operation is based on this assumption and cannot normalize the result for unrestricted values of A and B.

MNEMONIC: NRSQR

STATUS

N: The sign of the product.

Z: Zero flag is set when either A or B is zero (i.e. the exponent = 0 and the sign = 0). Sign and exponent of the product are set to zero.

COUT/U: Underflow flag is set when the exponent of the product is zero or negative. Sign and exponent of the product are set to zero.

OVER: Overflow flag is set when the exponent of the product is larger than FF hex.

STEX: Divide by Zero. This status bit is set when A or B are zero (i.e. exponent is zero). This will be the case when the exponent of B is 0 hex or 1 hex. The first case is a divide-by-zero and the second causes an intermediate overflow for which the seed rom sends out a zero exponent on A, which sets this flag.

INSTRUCTION: LS A * B OPCODE : MI = 8 hex

DESCRIPTION : Multiply Integer Single precision. The output is the least significant 32 bits of the 64 bit product of A and B. A and B are assumed to be signed integer and so is the result. The exponent is unpredictable.

MNEMONIC: MIS

STATUS

N: The sign of the product.

Z: Zero flag is set when either A or B is zero (i.e. A31-A0=0 or B31-B0=0).

COUT/U: Underflow flag is set to '0'.

OVER: Overflow flag is set when bits 66 through 31 of the product is neither all '0' nor all '1'.

STEX: Divide-by-Zero flag is set to '0'.



PROGRAMMING INFORMATION

INSTRUCTION: MS A * B OPCODE : MI = 9 hex

DESCRIPTION : Multiply Integer Double precision. The output is the most significant 32 bits of the 64-bit product of A and B. A and B are assumed to be signed integers and so is the result. The exponent is unpredictable.

MNEMONIC: MID

STATUS

- N:** The sign of the product.
- Z:** Zero flag is set when either A or B is zero (i.e. A31-A0=0 or B31-B0=0).
- COUT/U:** Overflow flag is set to '0'.
- OVER:** Underflow flag is set to '0'.
- STEX:** Divide-by-Zero flag is set to '0'

INSTRUCTION: LS A * #B OPCODE : MI = A hex

DESCRIPTION : Multiply Integer Single precision with B Unsigned. The output is the least significant 32 bits of the 64-bit product of A and B. A is assumed to be signed integer and B is assumed to be an unsigned number. The result is a signed integer. The exponent is unpredictable.

MNEMONIC: MISBU

STATUS

- N:** The sign of the product.
- Z:** Zero flag is set when either A or B is zero (i.e. A31-A0=0 or B31-B0=0).
- COUT/U:** Overflow flag is set when bits 66 through 31 of the product is neither all '0' nor all '1'.
- OVER:** Underflow flag is set to '0'.
- STEX:** Divide-by-Zero flag is set to '0'.

INSTRUCTION: MS A * #B OPCODE : MI = B hex

DESCRIPTION : Multiply Integer Double precision with B Unsigned. The output is the most significant 32 bits of the 64-bit product of A and B. A is assumed to be signed integer and B is assumed to be an unsigned number. The result is a signed integer. The exponent is unpredictable.

MNEMONIC: MIDBU

STATUS

- N:** The sign of the product.
- Z:** Zero flag is set when either A or B is zero (i.e. A31-A0=0 or B31-B0=0).
- COUT/U:** Underflow flag is set to '0'.
- OVER:** Overflow flag is set to '0'.
- STEX:** Divide-by-Zero flag is set to '0'.

INSTRUCTION: LS #A * #B OPCODE : MI = C hex

DESCRIPTION : Multiply Integer Single precision with A and B Unsigned. The output is the least significant 32 bits of the 64-bit product of A and B. A and B are assumed to be unsigned integers and so is the result. The exponent is unpredictable.

MNEMONIC: MISU

STATUS

- N:** The sign of the product, is zero.
- Z:** Zero flag is set when either A or B is zero (i.e. A31-A0=0 or B31-B0=0).
- COUT/U:** Underflow flag is set to '0'.
- OVER:** Overflow flag is set when bits 66 through 32 of the product is neither all '0' nor all '1'.
- STEX:** Divide-by-Zero flag is set to '0'.

PROGRAMMING INFORMATION

INSTRUCTION: MS #A * #B OPCODE : MI = D hex

DESCRIPTION : Multiply Integer Double precision with A and B Unsigned. The output is the most significant 32 bits of the 64-bit product of A and B. A and B are assumed to be unsigned integers and so is the result. The exponent is unpredictable.

MNEMONIC: MIDU

STATUS

- N:** The sign of the product, is zero.
- Z:** Zero flag is set when either A or B is zero (i.e. A31-A0=0 or B31-B0=0).
- COUT/U:** Underflow flag is set to '0'.
- OVER:** Overflow flag is set to '0'.
- STEX:** Divide-by-Zero flag is set to '0'.

INSTRUCTION: LS M(A) * M(B) OPCODE : MI = E hex

DESCRIPTION : Multiply Integer Single precision with A and B interpreted as floating-point mantissas. The output is the least significant 32 bits of the 64-bit product of A and B. A and B are assumed to be mantissas from floating-point numbers in the internal 40-bit floating-point format. The hidden bit is inserted and the sign bit is ignored by this instruction. No normalization is performed. The exponent is unpredictable.

MNEMONIC: MISM

STATUS

- N:** The sign of the product, is zero.
- Z:** Zero flag is set when either A or B is zero (i.e. A31-A0=0 or B31-B0=0 before the hidden bit is inserted into bit 31).
- COUT/U:** Underflow flag is set to '0'.
- OVER:** Overflow flag is set to '0'.
- STEX:** Divide-by-Zero flag is set to '0'.

INSTRUCTION: MS M(A) * M(B) OPCODE : MI = F hex

DESCRIPTION : Multiply Integer Double precision with A and B interpreted as floating-point mantissas. The output is the most significant 32 bits of the 64-bit product of A and B. A and B are assumed to be mantissas from floating-point numbers in the internal 40-bit floating-point format. The hidden bit is inserted and the sign bit is ignored by this instruction. No normalization is performed. The exponent is unpredictable.

MNEMONIC: MIDM

STATUS

- N:** The sign of the product, is zero.
- Z:** Zero flag is set when either A or B is zero (i.e. A31-A0=0 or B31-B0=0 before the hidden bit is inserted into bit 31).
- COUT/U:** Underflow flag is set to '0'.
- OVER:** Overflow flag is set to '0'.
- STEX:** Divide-by-Zero flag is set to '0'.

PROGRAMMING INFORMATION

Newton-Raphson 2 - A * B

The multiplier instruction NRDIV performs the first half of Newton-Raphson division cycles. A cycle of Newton-Raphson division consists of evaluating the equation $X * (2 - B * X)$ where B is the divisor and X is a close approximation of $1/B$. This is performed in two steps:

1. $2 - OPA * OPB$: where $OPA = X$ and $OPB = B$
2. $OPA * OPB$: where $OPA = X$ and $OPB =$ the result of step 1

Step two is just a standard floating-point multiplication and will not be discussed further. Step one is the special NRDIV instruction that requires further description to define its operation and to understand fully its correct use and results. These steps are repeated until a quotient of sufficient accuracy is obtained.

The NRDIV instruction can be used to perform the floating-point division of A/B for all possible values of A and B with certain restrictions such as overflow cases. This instruction will not function correctly for all possible values of input operands OPA and OPB. More specifically, the subtraction of the product $OPA * OPB$ from the quantity 2. This is acceptable since the desired operation is to perform the division of A/B and the restrictions on this individual instruction do not restrict the allowable values of A and B. The restrictions for the NRDIV instruction are added to simplify its hardware implementation. To allow a general $2 - A * B$ instruction would have required the addition of a complete normalization stage to the multiplier, thereby slowing its operation and increasing its die size.

The restriction on the operands to the NRDIV instruction is that one operand must be approximately the reciprocal of the other operand. It does not matter which operand is the smaller operand, only that they are reciprocals of each other. This restricts the result of their product to be approximately 1. If the on device seed (see Table 6) is used to generate a seed value for an approximate reciprocal of the divisor, then this condition will be fulfilled for all legal values of the divisor (values with a biased exponent of 0 or 1 are not legal values and will cause an overflow condition. The accuracy of the seed generated is approximately 8 bits and will produce a product that is more than sufficiently close to 1 to allow proper operation of this instruction. The minimum accuracy required for the reciprocal seed to function properly is approximately 2 significant binary bits. Seeds generated by other methods must at least meet this requirement for the NRDIV instruction to operate properly.

The need for this restriction is not in the multiplication part but in the subtraction part of the instruction. The multiplication of any operands A and B will require a mantissa normalization of at most one bit. Therefore, most multipliers do not contain normalization circuitry that will handle more than this one bit shift of the mantissa. A subtraction of any general operand A and B may require multiple bit shifts of the mantissa both to align the radix points initially and normalize the result afterwards. With the restriction on the input operands as above and knowing that the subtraction will always be from the fixed constant 2, the operation is constrained to require no more than a one bit shift of the mantissa result. The result of the product will be approximately 1, either slightly above or slightly below, and the subtraction from 2 will result in a quantity slightly below or slightly above 1, respectively. Normalization will require at most a one-bit shift as is already available in the multiplier.

The implementation of the actual subtraction takes advantage of one more observation that requires no further restrictions on the operands but allows further simplification of the hardware. The subtraction implementation eliminates the need for a hard-coded constant 2 by observing that a 2s complement on the mantissa performs exactly the same operation. This is evident after examination of appropriate bit strings.



PROGRAMMING INFORMATION

double-precision multiplication and addition

Double-precision (64-bit) operations are supported in the multiplier through eight fixed-point instructions that allow output of the upper or lower half of a 64-bit result. The ALU supports double-precision through instructions that allow multiple-precision fixed-point addition and 2s complement operations with carries propagated through an internal carry flip flop. The conversion and shift operations of the ALU are primarily single precision (32-bit) but also have application in double-precision functions. Floating-point double-precision operations may be done by decomposition into these single-precision fixed-point operations. Status will need to be selected from the results of the individual single-precision operations when available or specially generated by extra microcode steps. The select function of the ALU will aid in developing branchless microcode to handle exceptions.

Double-precision addition and subtraction of fixed-point values is done with the use of a carry flip-flop (CFF). The instructions that use this carry are:

1. $A + B + CFF \quad A1 = 09 \text{ hex}$
2. $A + \bar{B} + CFF \quad A1 = 0B \text{ hex}$
3. $\bar{A} + CFF \quad A1 = 11 \text{ hex}$

A double-precision or multiple-precision addition operation begins with a normal add with carryin such as $A + B + CIN$ on the lowest order 32-bit section of the operands. The next higher 32-bit sections are then added with instruction 1 above. Subsequent sections are also added using instruction 1. Instruction 2 would be used for subtraction and instruction 3 for one or 2s complement. The carry flip-flop is set by all of the integer arithmetic operations. The low-order addition must be performed immediately before the high-order addition in order to propagate the carry properly.

Double-precision multiplication is a composite of multiplication and addition operations. The double-precision operands must be stored separately as high and low values in the register file. The 32-bit integer field of the register holds the data. The exponent field is unused. A partial product algorithm is then used to multiply the high and low operand values and sum the outputs. For unsigned operands, this process is quite standard and does not need to be discussed further. Status must be determined by extra microcode steps as appropriate for the needs of the user's algorithm.

For signed double-precision multiplication, each operand may be treated by the multiplier as an unsigned fixed-point value, then correction cycles in the ALU would be used to correct the result. This method of signed multiplication that will implicitly extend the values by the sign is outlined as follow:

Double-precision integer:

$$A = |S| \dots \dots \dots 63 \text{ bits} \dots \dots \dots | = 2^{+63} + | \dots \dots \dots 63 \text{ bits} \dots \dots \dots |$$

$\xleftarrow{\text{2s Complement}} \qquad \qquad \qquad \xleftarrow{\text{Unsigned}}$

Implicit sign extended double-precision integer:

$$A = |S| \dots \dots \dots 63 \text{ bits} \dots \dots \dots | = -2^{+64} + | \dots \dots \dots 63 \text{ bits} \dots \dots \dots |$$

$\xleftarrow{\text{2s Complement}} \quad \text{(not stored)} \quad \xleftarrow{\text{Unsigned}}$

(S bit not stored)

$$= -2^{+64} + 2^{+63} + | \dots \dots \dots 63 \text{ bits} \dots \dots \dots |$$

$$= -2^{+63} + | \dots \dots \dots 63 \text{ bits} \dots \dots \dots |$$

If A and B are 2s complement 64-bit integers and UA and UB are the unsigned interpretation of these numbers as shown above, then the operations to be done in double precision are:

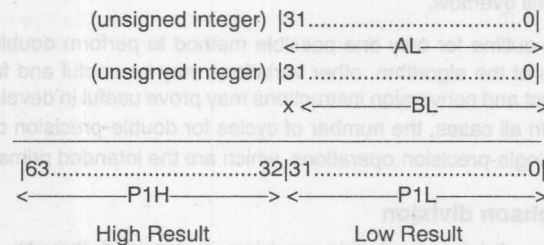
$$A * B = (-2^{+64} + UA) * (-2^{+64} + UB)$$

$$= (UA * UB) - A - B$$

PROGRAMMING INFORMATION

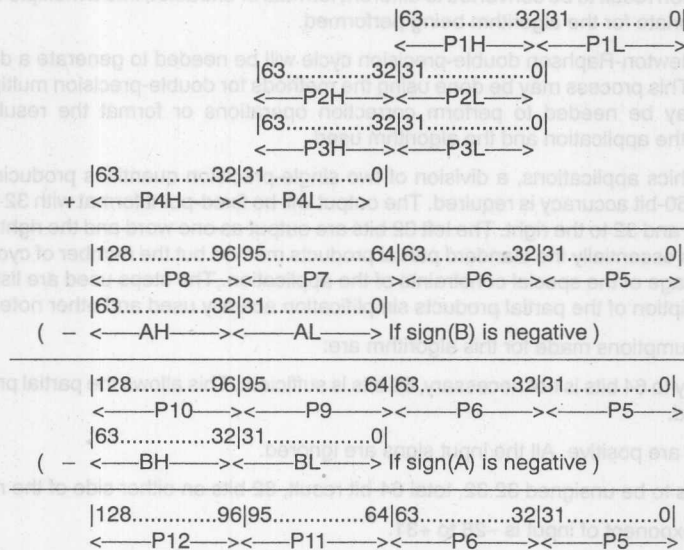
Each portion of each implicitly unsigned operand is multiplied by both portions of the other operand. The high- and low-order results of each multiplication are stored in the register file. Each portion is then added to corresponding pieces of the same magnitude in the ALU to produce a 128-bit result. This is essentially the unsigned double-precision function mentioned previously. Then the implicit sign of A is multiplied by B (high and low portions) and subtracted from the result by a conditional subtract operation. The implicit sign of B is also multiplied by A and subtracted to complete the double-precision multiply. The steps involved are shown below (AH,AL,BH,BL stand for A: high-order part, A: low-order part, B: high-order part, and B: low-order part, respectively):

Step 1-4: AL*BL, AL*BH, AH*BL, AH*BH



Now the partial product results must be summed with other partial products of the same magnitude:

Step 5:



The conditional correction steps could be performed by branches in microcode or by using the select instruction of the ALU to select the adjusted or unadjusted quantity. The selection could be performed by shifting the sign bit of A or B into the LFF and feeding this status output back to the CIN pin to control the selection.

PROGRAMMING INFORMATION

If the input operands are interpreted as integers, then the binary point of the result is at the extreme right. The binary point in the final result would be located between bits 127 and 126 for fractional operands. Other interpretations of the input operands would result in different interpretations of the result and status as appropriate. A desired section of the 128-bit string could be selected through use of the shift and logical operations in the ALU if desired.

As mentioned, status for the double-precision operations will depend on the interpretation of the operands and may require extra steps to check the results. In general, status from each individual step will not be very useful, but status generated by the final multiply or addition steps may provide part of the needed information. For example, overflow when from the multiplication of the low-order pieces in double-precision does not necessarily indicate the entire operation will overflow.

This description provides the outline for only one possible method to perform double-precision operations. Depending on the constraints of the algorithm, other variations may be useful and faster. For floating-point operations the exponent extract and conversion instructions may prove useful in developing relatively efficient double-precision algorithms. In all cases, the number of cycles for double-precision operations will be many more times than needed for single-precision operations, which are the intended primary usage of the device.

double-precision Newton-Raphson division

Support for Newton-Raphson division in double-precision is provided through use of multiple cycle double-precision multiplication and addition operations outlined above. The seed used for double-precision is typically the result of a single-precision floating-point division. The conversion instructions of the ALU will allow the single-precision result to be converted to different formats or extracted into a multiple word double-precision format as appropriate for the algorithm being performed.

One additional Newton-Raphson double-precision cycle will be needed to generate a double-precision result for most needs. This process may be done using the methods for double-precision multiplication and addition. Other cycles may be needed to perform correction operations or format the results depending on the requirements of the application and the algorithm used.

For certain graphics applications, a division of two single-precision quantities producing a double-precision value of at least 50-bit accuracy is required. The output will be fixed-point format with 32 binary digits to the left of the radix point and 32 to the right. The left 32 bits are output as one word and the right as another word. The algorithm used is essentially the standard partial products method but the number of cycles has been reduced by taking advantage of the special constraints of the application. The steps used are listed in Table 25, which includes a description of the partial products simplification actually used and other notes.

The general assumptions made for this algorithm are:

- Accuracy to 64 bits is not necessary, 60 bits is sufficient. This allows the partial product procedure to be simplified.
- A and B are positive. All the input signs are ignored.
- Output is to be unsigned 32.32, total 64-bit result, 32 bits on either side of the radix point.
- Binary exponent of input is -28 to +31.
- Status can be obtained from the single-precision steps or will be generated separately.

PROGRAMMING INFORMATION

Table 26. Register File Allocation

| REGISTER | VARIABLE | REGISTER | VARIABLE |
|----------|------------|----------|----------|
| RF00 | A | RF08 | PH |
| RF01 | B | RF09 | SH1, SHF |
| RF02 | X0, X1, X2 | RF0A | SH2, SHM |
| RF03 | CL, SPL | RF0B | — |
| RF04 | CH, SPH | RF0C | 255 |
| RF05 | DL | RF0D | — |
| RF06 | DH | RF0E | 1 |
| RF07 | PL | RF0F | 0 |

Partial Products:

The double-precision cycle completes the equation:

$$A \times X2 \times (2 - (B \times X2))$$

This requires two 32-bit by 32-bit multiplications, one 64-bit 2s complement, and one 64-bit by 64-bit multiplication as follows:

A * X2

Steps: M8,M9 — A and X2 are the 32-bit mantissas from the single-precision NR division. These steps insert the hidden bit and multiply saving the high and low 32 bits of the result. The result is not normalized nor is the leading 1 made into a hidden bit.

B * X2

Steps: M6,M7 — B and X2 are the 32-bit mantissas from the single-precision NR division. These steps insert the hidden bit and multiply saving the high and low 32 bits of the result. The result is not normalized nor is the leading 1 made into a hidden bit.

2 - B * X2

Steps: A7,A8 — The 2 minus step is accomplished by 2s complementing the 64-bit B * X2 product. This can be done since B * X2 is close to 1.

A * X2 * (2 - (B * X2))

Steps: M10,M11,M12,M13 multiply the high and low 32-bit pieces of the two 64-bit quantities A * X2 and 2 - (B * X2) to produce the high 64 bits of the 128-bit result. The low 64 bits of the 128-bit result are not needed and missing carry outs will not affect the required accuracy in the high 64 bits. Again, an accuracy of 60 bits is sufficient for this application.

Steps: A12,A13,A14,A15 add the pieces of the partial product that contribute to the high 64 bits of the result.

PROGRAMMING INFORMATION

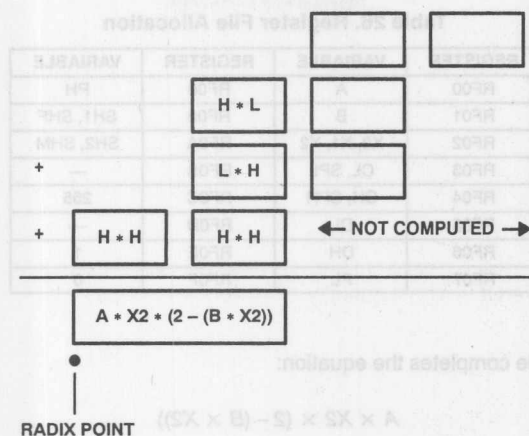


Figure 10. Results of Steps

The result is an unnormalized 64-bit fixed-point quantity with the radix point to the extreme left. Formatting for output consists of shifting this quantity to the right according to the shift quantity computed from the exponents of A and X2 in steps A4, A5, A6, A9, and A10. The actual shifting is performed in the multiplier by converting the shift count to a power of 2. Steps M15, M16, M17, A16, A17, A18, and A19 perform the shift needed.

double-precision Newton-Raphson division cycle support

$\text{SINGLE}(A)/\text{SINGLE}(B) = \text{DOUBLE}(Q)$

Cycle 1: Load A and B

MULT: idle

ALU: idle

A is loaded through DA and stored in RF00; B is loaded through DB and stored in RF01. Add one more cycle if A and B must be converted from fixed-point to floating-point format. Add another cycle, if A and B will be loaded only through a single port.

single-precision mode

Cycle 2: NR single-precision division $2 - X0 * B$

MULT: NRDIV(SEED, RF01) \Rightarrow MULT B

ALU: idle

First half of initial NR cycle. Load constant value 255 into the register file. The seed value is generated by applying RF01 to the seed (see Table 6).

Cycle 3: NR single-precision division $X0 (2 - X0 * B) \Rightarrow X1$

MULT: SEED * MULT B \Rightarrow MULT B, RF02

ALU: idle

Second half of initial NR cycle. Load constant values 1 and 0.

PROGRAMMING INFORMATION

Cycle 4: NR single-precision division $2 - B * X1$

MULT: NRDIV(RF01, MULT B) \Rightarrow MULT B

ALU: E2I(A) \Rightarrow ALU B

First half of second NR cycle. Extract exponent of A for later shift operation.

Cycle 5: NR single-precision division $X1 (2 - B * X1) \Rightarrow X2$

Add EXP(A) - 255 \Rightarrow SH1

MULT: RF02 * MULT B \Rightarrow MULT B, ALU A, RF02

ALU: -255 + ALU B \Rightarrow RF09

Second half of second NR cycle. Result from MULT is accurate to approximately 31 bits. Constant is subtracted from the exponent to convert it to a shift quantity.

double-precision cycle

Cycle 6: Low half of mantissa product $B * X2$

EXP(X2) \Rightarrow SH2

MULT: MISM(RF01, MULT B) \Rightarrow ALU A

ALU: E2I(ALU A) \Rightarrow RF0A

Insert the hidden bit and multiply the mantissas of B and X2 selecting the low-order 32 bits. Extract the exponent of X2 for later shift operation.

Cycle 7: High half of mantissa product $B * X2$

Complement low half of $B * X2 \Rightarrow$ CL

MULT: MIDM(RF01, RF02) \Rightarrow ALU A

ALU: NOT(ALU A) + CIN \Rightarrow RF03

Insert the hidden bit and multiply the mantissas of B and X2 selecting the high-order 32 bits. Complement low half of $B * X2$, CIN=1.

Cycle 8: Low half of mantissa product $A * X2 \Rightarrow$ DL

Complement high half of $B * X2 \Rightarrow$ CH

MULT: MISM(RF00, RF02) \Rightarrow RF05

ALU: NOT(ALU A) + CFF \Rightarrow RF04

Insert the hidden bit and multiply the mantissas of A and X2 selecting the low-order 32 bits. Complement high half of $B * X2$.

Cycle 9: High half of mantissa product $A * X2 -$ DH

Add EXP(A) + EXP(X2) - 255 \Rightarrow SHF

MULT: MIDM(RF00, RF02) \Rightarrow RF06

ALU: RF09 + RF0A \Rightarrow ALU B, RF09

Insert the hidden bit and multiply the mantissas of A and X2 selecting the high-order 32 bits. Finish adding exponents and constant for shift value (SH1 + SH2 - 255).

Cycle 10: High piece of partial product $CH * DL \Rightarrow$ PL1

Shift a 1 left by SHF \Rightarrow SHM

MULT: HIGH(RF04 * RF05) \Rightarrow RF07

ALU: SLL(1, ALU B) \Rightarrow RF0A

Multiply the partial product to get first piece of the low half. Convert the shift factor (SHF) to a power of 2 to use the multiplier for shifting.

PROGRAMMING INFORMATION

Cycle 11: High piece of partial product $CL * DH \Rightarrow PL2$

MULT: HIGH(RF03 * RF06) \Rightarrow ALU A

ALU: idle

Multiply the partial product to get second piece of the low half.

Cycle 12: High piece of partial product $CH * DH \Rightarrow PH$

Add low pieces $PL2 + PL1 \Rightarrow PL$

MULT: HIGH(RF04 * RF06) \Rightarrow ALU A

ALU: ALU A + RF07 \Rightarrow RF07

Multiply the partial product to get the only piece of the high half. Add the first two low pieces of the partial product with a carry out.

Cycle 13: Low piece of partial product $CH * DH \Rightarrow PL3$

Add high piece and carry $PH + CFF \Rightarrow PH$

MULT: LOW(RF04 * RF06) \Rightarrow ALU A

ALU: ALU A + CFF \Rightarrow RF08

Multiply the partial product to get last piece of the low half. Add the carry out from the low piece to the high piece.

Cycle 14: Add last low piece $PL3 + PL \Rightarrow PL$

MULT: idle

ALU: ALU A + RF07 \Rightarrow MULT A, RF07

Complete the addition of the three low pieces of the partial product with a carry out.

Cycle 15: Shift low partial product $PL * SHM \Rightarrow SPL$

Add high piece and carry $PH + CFF \Rightarrow PH$

MULT: HIGH(MULT A * RF0A) \Rightarrow RF03

ALU: RF08 + CFF \Rightarrow MULT A, RF08

Use the multiplier as a simple shifter to shift the low result of the partial product for output formatting. Add the carry out to the high result of the partial product.

Cycle 16: Shift high partial product $PH * SHM \Rightarrow SPM$

Rotate left the shift factor $ROL(SHF) \Rightarrow LFF$

MULT: LOW(MULT A * RF0A) \Rightarrow ALU A

ALU: $ROL(RF09) \Rightarrow LFF$

Multiplier shift of high partial product to obtain the fill bits for the shifted low partial product. Rotate the shift value left in the ALU to set the LFF. This clears the LFF, if the shift value was positive or zero; and sets the LFF, if it was negative.

Cycle 17: Shift high partial product $PH * SHM \Rightarrow SPH$

Combine low piece and fill bits $OR(SPM, SPL) \Rightarrow SPL$

MULT: HIGH(RF08 * RF0A) \Rightarrow ALU A, RF04

ALU: $OR(ALU A, RF03) \Rightarrow RF03$

Multiplier shift the high partial product to obtain the high result. OR the low result with the fill bits to obtain the low result. The OR operation does not destroy the setting of the LFF.



PROGRAMMING INFORMATION

Cycle 18: Select high result for output SEL(SPH, 0) ⇒ Y

MULT: idle

ALU: SEL(ALU A, RF0F) ⇒ SUM

Select the result for the high half of the result (integer part) based on the setting of the LFF. The shift value being negative indicates the result must be shifted an extra 32 bits to the right so the high result should be all zeroes.

Cycle 19: Select low result for output SEL(SPL, SPH) ⇒ Y

Output high result on Y

MULT: idle

ALU: SEL(RF03, RF04) ⇒ SUM

Select the result for the low half of the result (fraction part) based on the setting of the LFF. The shift value being negative indicates the result must be shifted an extra 32 bits to the right, so the low result should be the high half. The high result will now be available at the Y port.

Cycle 20: Output high result on Y

MULT: idle

ALU: idle

The low result will now be available at the Y port.

| INSTRUCTION | DATA | STATUS |
|-------------|-------|--------|
| 000 | A + A | 000 |
| 001 | A - A | 001 |
| 010 | A + B | 010 |
| 011 | A - B | 011 |
| 100 | A + C | 100 |
| 101 | A - C | 101 |
| 110 | A + D | 110 |
| 111 | A - D | 111 |
| 000 | A + E | 000 |
| 001 | A - E | 001 |
| 010 | A + F | 010 |
| 011 | A - F | 011 |
| 100 | A + G | 100 |
| 101 | A - G | 101 |
| 110 | A + H | 110 |
| 111 | A - H | 111 |
| 000 | A + I | 000 |
| 001 | A - I | 001 |
| 010 | A + J | 010 |
| 011 | A - J | 011 |
| 100 | A + K | 100 |
| 101 | A - K | 101 |
| 110 | A + L | 110 |
| 111 | A - L | 111 |

PROGRAMMING INFORMATION

ALU instruction set

ALU operations are under control of instruction bits A14-A10. Bits 3 and 4 signify integer or floating-point operations as described in Table 27.

Table 27. ALU Instruction Set

| INSTRUCTION BITS | | ALU OPERATIONS |
|------------------|---------|---------------------------------------|
| A14-A13 | A12-A10 | |
| 00† | 000 | A + B |
| | 001 | A - B |
| | 010 | Compare A, B (using A - B) |
| | 011 | Pass A |
| | 100 | Pass (-A) |
| | 101 | Floating-Point to Integer Conversion |
| | 110 | Floating-Point to Fraction Conversion |
| | 111 | Integer to Exponent Conversion |
| 01§ | 000 | A + B + CIN |
| | 001 | A + B + CFF‡ |
| | 010 | A + B̄ + CIN |
| | 011 | A + B̄ + CFF‡ |
| | 100 | Ā + B + CIN |
| | 101 | Pass A/B on CIN |
| | 110 | MIN/MAX on CIN |
| | 111 | A + CIN |
| 10§ | 000 | Ā + CIN |
| | 001 | Ā + CFF‡ |
| | 010 | A AND B |
| | 011 | Ā AND B |
| | 100 | A XOR B |
| | 101 | A OR B |
| | 110 | A NAND B |
| | 111 | A NOR B |

† Floating-point format

‡ The carry flip-flop CFF retains the carry-out status COUT of the previous ALU result for one cycle.

§ Integer/Logical format

PROGRAMMING INFORMATION

Table 27. ALU Instruction Set (Continued)

| INSTRUCTION BITS | | ALU OPERATIONS |
|------------------|---------|--|
| A14-A13 | A12-A10 | |
| 1 0 [§] | 000 | Shift Right Logical A (clear L Flip-Flop) [†] |
| | 001 | Shift Right Arithmetic A |
| | 010 | Rotate Right B [¶] |
| | 011 | Rotate Left B [¶] |
| | 100 | Shift Left Logical A (set L Flip-Flop) [†] |
| | 101 | Exponent to integer conversion |
| | 110 | Integer to floating-point conversion |
| | 111 | Fractional to floating-point conversion |

[†] The carry flip-flop CFF retains the carry-out status COUT of the previous ALU result for one cycle.

[‡] SRL/SLL — Barrel shift capability (shift count at B port).

[§] Integer/Logical format

[¶] ROR/ROL — Rotate left or rotate right by one bit. The bit lost is stored in the link flip-flop. The bit gained is output from the link flip-flop (see Figure 12).

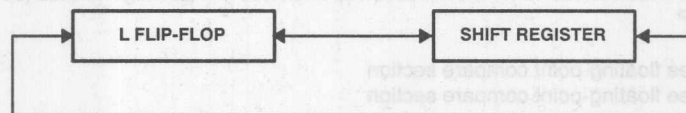


Figure 11. Shift Register and Link Flip-Flop

The following text gives a more complete description of each of the ALU opcodes and their operation. The general format used is:

STATUS N: Sign
Z: Zero
COUT/U: Underflow
OVER: Overflow
STEX: Divide-by-Zero

INSTRUCTION: A + B OPCODE : A1 = 0 hex

DESCRIPTION: Addition of floating-point operands. The result is a 31-bit mantissa (not rounded), sign bit, and an 8-bit exponent in internal single precision format.

MNEMONIC: ADDF

STATUS

N: Sign of the result.

Z: Set when the result is zero.

COUT/U: Underflow occurs when, in order to normalize the mantissa after the addition operation, the biased exponent is less than 1. The result output from the ALU on underflow is a floating-point zero.

OVER: Overflow occurs when, in order to normalize the mantissa after the addition operation, the biased exponent exceeds 255 decimal. During an overflow, the resulting output from the ALU has the correct mantissa and sign. However, the exponent wraps around.

INSTRUCTION: A - B OPCODE : AI = 1 hex

DESCRIPTION : Subtraction of floating-point operands. The result is a 31-bit mantissa (not rounded), sign bit, and an 8-bit exponent in internal single precision format.

MNEMONIC: SUBF

STATUS

- N:** Sign of the result.
Z: Set when the result is zero.
COUT/U: Underflow occurs when in order to normalize the mantissa after the subtraction operation the biased exponent is less than 1. The result output from the ALU during an underflow is a floating point zero.
OVER: Overflow occurs when in order to normalize the mantissa after the subtraction operation the biased exponent exceeds 255. The result output from the ALU during an overflow has the correct mantissa and sign. However, the exponent wraps around.

INSTRUCTION: COMPARE A,B OPCODE : AI = 2 hex

DESCRIPTION : Compares two floating-point operands and generates two status bits (N,Z). The operation $A - B$ is used. The status can be evaluated identically to status from an integer subtraction to determine A GT B, A GE B, A EQ B, A LT B, and A LE B. The result output from the ALU is that of a floating-point subtraction operation. This instruction varies from the subtraction operation only in status generated (see Tables 23 and 24).

MNEMONIC: CMP

STATUS

- N:** see floating-point compare section
Z: see floating-point compare section
COUT/U: 0
OVER: 0
STEX:

EXAMPLE : RAA = H'01F7654321' RAB = H'01F7654321'
 Result = H'xxxxxxxx' Status = B'0100'
 RAA = H'01F7654321' RAB = H'0177654321'
 Result = H'xxxxxxxx' Status = B'1000'

INSTRUCTION: PASS A OPCODE : AI = 3 hex

DESCRIPTION : Pass floating-point data on RAA input to the sum bus. The operand on the RAB input is ignored by this instruction.

MNEMONIC: PA

STATUS

- N:** Sign of A.
Z: Set when A is zero.
COUT/U: 0
OVER: 0



PROGRAMMING INFORMATION

INSTRUCTION: PASS (-A) OPCODE : AI = 4 hex

DESCRIPTION : Pass floating-point data on RAA input to the sum bus while inverting the sign bit. The operand on the RAB input is ignored by this instruction.

MNEMONIC: PNA

STATUS

N: Inverted sign of A.
Z: Set when A is zero.

COUT/U: 0

OVER: 0

INSTRUCTION: FLOAT TO INTEGER CONVERSION OPCODE : AI = 5 hex

DESCRIPTION : Converts a floating-point internal F format number on the RAA port to a signed integer with no rounding. The operand on the RAB port is ignored by this instruction. The resulting integer has the eight exponent bits set to 0.

MNEMONIC: F2I

STATUS

N: Sign of the result.
Z: Set when the result is zero.

COUT/U: 0. Underflow actually occurs when the biased exponent is less than 129. Although the floating point operand is too small to be represented, its truncated fractional value is equal to an integer zero. The result output on the sum bus is therefore a integer zero, and no underflow status is set.

OVER: Overflow occurs when the biased exponent is larger than 160 decimal (except for the case where the exponent is 161 decimal, the sign negative, and the explicit mantissa is all zero). Therefore, the floating-point operand is too large to be represented in integer format. The result output on the sum bus is the 2s complement value of the mantissa unshifted.

INSTRUCTION: FLOAT TO FRACTION CONVERSION OPCODE : AI = 6 hex

DESCRIPTION : Converts a floating-point internal F format number on the RAA port to a signed integer with radix point to the right of the MSB and with no rounding. The operand on the B port is ignored by this instruction. The resulting fraction has the eight exponent bits set to 0.

MNEMONIC: F2FR

STATUS

N: Sign of the result.
Z: Set when the result is zero.

COUT/U: 0. Underflow occurs when the biased exponent is less than 97 decimal. Although the floating point operand is too small to be represented, its truncated fractional value is equal to a fractional zero (0.0...0). The result output on the sum bus is therefore an integer zero, and no underflow status is set.

OVER: Overflow occurs when the biased exponent is larger than 128 decimal (except for the case where the exponent is 129 decimal, the sign negative, and the explicit mantissa is all zero). Therefore, the floating-point operand is too large to be represented in signed fractional format. The result output on the SUM bus is the 2s complement value of the mantissa unshifted.

PROGRAMMING INFORMATION

INSTRUCTION: INTEGER TO EXPONENT CONVERSION OPCODE : AI = 7 hex

DESCRIPTION : Takes the least significant 8 bits of operand RAA and inserts them into the exponent field of operand RAB, thereby replacing the operand RAB'S exponent. The mantissa and sign bit of operand RAB are left unaltered.

MNEMONIC: I2E

STATUS

N: Sign of the result.
 Z: Set when the result is zero.

COUT/U: 0

OVER: 0

STEX:

EXAMPLE : RAA = H'000FED7B98' RAB = H'83800154A7'
 Result = H'98800154A7' Status = B'1000'

RAA = H'00AC701234' RAB = H'FF800154A7'
 Result = H'34800154A7' Status = B'1000'

INSTRUCTION: A + B + CIN OPCODE : AI = 8 hex

DESCRIPTION : Integer addition of operand RAA to RAB with a carry-in from CIN pin.

MNEMONIC:

STATUS

N: Set normally as required.

Z: Set normally as required.

COUT/U: Set normally as required.

OVER: Set normally as required.

INSTRUCTION: A + B + CFF OPCODE : AI = 9 hex

DESCRIPTION : Integer add of RAA to RAB with a carry-in from carry flip-flop. The carry flip-flop is enabled for each integer arithmetic operation.

MNEMONIC:

STATUS

N: Set normally as required.

Z: Set normally as required.

COUT/U: Set normally as required.

OVER: Set normally as required.

INSTRUCTION: A + \bar{B} + CIN OPCODE : AI = A hex

DESCRIPTION : Integer add of RAA to NOT RAB with a carry-in from CIN pin.

MNEMONIC:

STATUS

N: Set normally as required.

Z: Set normally as required.

COUT/U: Set normally as required.

OVER: Set normally as required.

STEX:

PROGRAMMING INFORMATION

INSTRUCTION: $A + \bar{B} + \text{CFF}$ OPCODE : AI = B hex

DESCRIPTION : Integer add of RAA to NOT RAB with a carry-in from carry flip-flop. The carry flip-flop is enabled for every integer arithmetic operation.

MNEMONIC:

STATUS

- N: Set normally as required.
- Z: Set normally as required.
- COUT/U: Set normally as required.
- OVER: Set normally as required.

INSTRUCTION: $\bar{A} + B + \text{CIN}$ OPCODE : AI = C hex

DESCRIPTION : Integer add of NOT RAA to RAB with a carry-in from CIN pin.

MNEMONIC:

STATUS

- N: Set normally as required.
- Z: Set normally as required.
- COUT/U: Set normally as required.
- OVER: Set normally as required.

INSTRUCTION: PASS A / B on CIN OPCODE : AI = D hex

DESCRIPTION : This instruction passes the A or B operand depending on the value of the CIN bit. If CIN = 0 then A is passed, if CIN = 1 then B is passed. This instruction passes both integer and floating-point operands.

MNEMONIC:

STATUS

- N: Set normally as required.
- Z: Set normally as required for integers (see Note 2)
- COUT/U: 0.
- OVER: 0.

CAUTION: THE STATUS REGISTER MUST BE ENABLED DURING THIS INSTRUCTION. This is necessary to avoid unstable conditions resulting from the status being fed back into the ALU control.

NOTE 2: If the operands being passed are floating-point, the pass operation will work correctly. However, the ZERO status may be incorrect. This instruction is part of the ALU integer instruction set and therefore for all operands being passed the ZERO status is determined as if the operand were an integer. For more details see floating-point status under status section.

INSTRUCTION: MIN / MAX on CIN OPCODE : AI = E hex

DESCRIPTION : This instruction passes the A or B operand depending on the registered ALU status bits of the previous operation. When CIN = 0, the minimum value is selected; when CIN = 1, the maximum value is selected. The comparison between the value of A and B is based on the ALU status bits N and OVER (see Table 22).

MNEMONIC:

STATUS

- N: Set normally as required.
- Z: Set normally as required for integers. (see Note 3)
- COUT/U: 0
- OVER: 0

CAUTION: STATUS REGISTER MUST BE ENABLED DURING THIS INSTRUCTION. This is necessary to avoid unstable conditions resulting from the status being fed back into the ALU control in this instruction.

NOTE 3: If the operands being passed are floating-point, the pass operation will work correctly. However, the ZERO status may be incorrect. This instruction is part of the ALU integer instruction set and therefore for all operands being passed the ZERO status is determined as if the operand were an integer. For more details see floating-point status under status section.

PROGRAMMING INFORMATION

INSTRUCTION: A + CIN OPCODE : AI = F hex

DESCRIPTION : Addition of RAA to zero with a carry-in from CIN pin.

MNEMONIC:

STATUS

N: Sign of the result.

Z: Set when the result is zero.

COUT/U: Carry out of the MSB.

OVER: Set normally as required.

INSTRUCTION: \bar{A} + CIN OPCODE : AI = 10 hex

DESCRIPTION : Integer addition of NOT RAA to zero with a carry-in from CIN pin. When CIN = 1, then this is the 2s complement of RAA which is equivalent to the (2-A) operation in the Newton-Raphson divide algorithm.

MNEMONIC:

STATUS

N: Sign of the result.

Z: Set when the result is zero.

COUT/U: Carry out of MSB.

OVER: Set normally as required.

INSTRUCTION: \bar{A} + CFF OPCODE : AI = 11 hex

DESCRIPTION : Integer addition of NOT RAA to zero with a carry-in from carry flip-flop. As long as CFF = 1, then this is the 2s complement of RAA which is equivalent to the (2 - A) operation in the Newton-Raphson divide algorithm, otherwise this is a NEGATE operation.

MNEMONIC:

STATUS

N: Sign of the result.

Z: Set when the result is zero.

COUT/U: Carry out of MSB.

OVER: Set normally as required.

INSTRUCTION: A AND B OPCODE : AI = 12 hex

DESCRIPTION : Logical AND of RAA and RAB.

MNEMONIC:

STATUS

N: Sign of the result.

Z: Set when the result is zero.

COUT/U: 0.

OVER: 0.

INSTRUCTION: \bar{A} AND B OPCODE : AI = 13 hex

DESCRIPTION : Logical AND of NOT RAA and RAB.

MNEMONIC:

STATUS

N: Sign of the result.

Z: Set when the result is zero.

COUT/U: 0

OVER: 0



PROGRAMMING INFORMATION

INSTRUCTION: A XOR B OPCODE : AI = 14 hex

DESCRIPTION : Exclusive OR of RAA and RAB.

MNEMONIC:

STATUS

N: Sign of the result.
Z: Set when the result is zero.

COUT/U: 0
OVER: 0

INSTRUCTION: A OR B OPCODE : AI = 15 hex

DESCRIPTION : Logical OR of RAA and RAB.

MNEMONIC:

STATUS

N: Sign of the result.
Z: Set when the result is zero.

COUT/U: 0
OVER: 0

INSTRUCTION: A NAND B OPCODE : AI = 16 hex

DESCRIPTION : Logical NAND of RAA and RAB.

MNEMONIC:

STATUS

N: Sign of the result.
Z: Set when the result is zero.

COUT/U: 0
OVER: 0

INSTRUCTION: A NOR B OPCODE : AI = 17 hex

DESCRIPTION : Logical NOR of RAA and RAB.

MNEMONIC:

STATUS

N: Sign of the result.
Z: Set when the result is zero.

COUT/U: 0
OVER: 0

INSTRUCTION: SHIFT RIGHT LOGICAL A OPCODE : AI = 18 hex

DESCRIPTION : Performs a barrel shift right on operand RAA. The shift count comes from the least significant 5 bits of the RAB operand. The more significant bits of operand RAB are not used. The shift quantity is always interpreted as a positive value. This instruction also clears the link flip-flop. Zeroes are filled in from the left.

MNEMONIC: SRL

STATUS

N: Sign of the result.
Z: Set when the result is zero.

COUT/U: 0
OVER: 0

EXAMPLE :

RAA = H'00A170258D' RAB = H'00AD9754C0'
Result = H'00A170258D' Status = B'1000'

PROGRAMMING INFORMATION

INSTRUCTION: SHIFT RIGHT ARITHMETIC A OPCODE : AI = 19 hex

DESCRIPTION : Performs a barrel shift right on operand RAA, filling with the sign bit. The shift count comes from the least significant 5 bits of the RAB operand. The shift quantity is always interpreted as a positive value. This instruction also clears the link flip-flop upon every execution. The result is sign filled on the left.

MNEMONIC: SRA

STATUS

N: Sign of the result.
Z: Set when the result is zero.

COUT/U: 0
OVER: 0

INSTRUCTION: ROTATE RIGHT B OPCODE : AI = 1A hex

DESCRIPTION : Rotates operand RAB to the right by one bit position. The most significant bit is filled with the contents of the link flip-flop, and the least significant bit lost during the rotation is stored in the link flip-flop on the next rising edge of the clock. The operand on the RAA port is ignored.

MNEMONIC: ROR

STATUS

N: Sign of the result.
Z: Set when the result is zero.

COUT/U: 0
OVER: 0

EXAMPLE :

RAA = H'xxxxxxxx' RAB = H'00A170258D' Link Flip-Flop = 0
Result = H'0050B812C6' Status = B'0000' Link Flip-Flop = 1

RAA = H'xxxxxxxx' RAB = H'0050B812C6' Link Flip-Flop = 1
Result = H'00A85C0963' Status = B'1000' Link Flip-Flop = 0

RAA = H'xxxxxxxx' RAB = H'00A85C0963' Link Flip-Flop = 0
Result = H'00542E04B1' Status = B'0000' Link Flip-Flop = 1

INSTRUCTION: ROTATE LEFT B OPCODE : AI = 1B hex

DESCRIPTION : Rotates operand RAB to the left one bit position. The least significant bit is filled with the contents of the link flip-flop, and the most significant bit lost during the rotation is stored in the link flip-flop on the very next rising edge of the clock. The operand on the RAA port is ignored by this instruction.

MNEMONIC: ROL

STATUS

N: Sign of the result.
Z: Set when the result is zero.

COUT/U: 0
OVER: 0

EXAMPLE :

RAA = H'xxxxxxxx' RAB = H'00542E04B1' Link Flip-Flop = 1
Result = H'00A85C0963' Status = B'1000' Link Flip-Flop = 0

RAA = H'xxxxxxxx' RAB = H'00A85C0963' Link Flip-Flop = 0
Result = H'0050B812C6' Status = B'0000' Link Flip-Flop = 1

RAA = H'xxxxxxxx' RAB = H'0050B812C6' Link Flip-Flop = 1
Result = H'00A170258D' Status = B'1000' Link Flip-Flop = 0



PROGRAMMING INFORMATION

INSTRUCTION: SHIFT LEFT LOGICAL A OPCODE : AI = 1C hex

DESCRIPTION : Performs a barrel shift left on operand RAA. The shift count comes from the least significant 5 bits of the RAB operand. The shift quantity is always interpreted as a positive value. This instruction also sets the link flip-flop upon every execution. Zeroes are filled from the right.

MNEMONIC: SLL

STATUS

N: Sign of the result.

Z: Set when the result is zero.

COUT/U: 0

OVER: 0

INSTRUCTION: EXPONENT TO INTEGER CONVERSION OPCODE : AI = 1D hex

DESCRIPTION : Takes the eight-bit exponent field of operand A and inserts them into the least significant eight bits of the output. All other bits are zeroed. The operand on the RAB input is ignored by this instruction.

MNEMONIC: E2I

STATUS

N: 0

Z: Set when the result is zero.

COUT/U: 0

OVER: 0

EXAMPLE :

RAA = H'84005D7900' RAB = H'xxxxxxxxxx'

Result = H'0000000084' Status = B'0000'

RAA = H'C9F876CC00' RAB = H'xxxxxxxxxx'

Result = H'00000000C9' Status = B'0000

INSTRUCTION: INTEGER TO FLOAT CONVERSION OPCODE : AI = 1E hex

DESCRIPTION : Converts an integer to a floating-point number. The operand on the RAB input is ignored by this instruction.

MNEMONIC: I2F

STATUS

N: Sign of the result.

Z: Set when the result is zero.

COUT/U: 0

OVER: 0

INSTRUCTION: FRACTIONAL TO FLOAT CONVERSION OPCODE : AI = 1F hex

DESCRIPTION : Converts an integer with radix point to the right of the MSB to a floating-point number. The operand on the RAB input is ignored by this instruction.

MNEMONIC: FR2F

STATUS

N: Sign of the result.

Z: Set when the result is zero.

COUT/U: 0

OVER: 0

PROGRAMMING INFORMATION

INSTRUCTION: SHIFT LEFT LOGICAL A (SLOD) : AI = 1D hex
DESCRIPTION: Performs a barrel shift left on operand RAA. The shift count comes from the least significant 5 bits of the RAB operand. The shift quantity is always interpreted as a positive value. This instruction also sets the link flip-flop upon every execution. Zeros are filled from the right.
MEMONIC: SL
STATUS:

N: Sign of the result.
 Z: Set when the result is zero.
 COUT: 0
 OVER: 0

INSTRUCTION: EXPONENT TO INTEGER CONVERSION (OPEC) : AI = 1D hex
DESCRIPTION: Takes the eight-bit exponent field of operand A and inserts them into the least significant eight bits of the output. All other bits are zeroed. The operand on the RAB input is ignored by this instruction.
MEMONIC: EI
STATUS:

N: 0
 Z: Set when the result is zero.
 COUT: 0
 OVER: 0

EXAMPLE:
 RAA = H'408D7800, RAB = H'00000000
 Result = H'00000000, Status = B'0000
 RAA = H'08873C00, RAB = H'00000000
 Result = H'00000000, Status = B'0000

INSTRUCTION: INTEGER TO FLOAT CONVERSION (OPEC) : AI = 1E hex
DESCRIPTION: Converts an integer to a floating-point number. The operand on the RAB input is ignored by this instruction.
MEMONIC: IF
STATUS:

N: Sign of the result.
 Z: Set when the result is zero.
 COUT: 0
 OVER: 0

INSTRUCTION: FRACTIONAL TO FLOAT CONVERSION (OPEC) : AI = 1F hex
DESCRIPTION: Converts an integer with radix point to the right of the MSB to a floating-point number. The operand on the RAB input is ignored by this instruction.
MEMONIC: FRF
STATUS:

N: Sign of the result.
 Z: Set when the result is zero.
 COUT: 0
 OVER: 0

| | |
|---|-----------|
| General Information | 1 |
| SN74ACT8818A 16-Bit Microsequencer | 2 |
| SN74ACT8832A 32-Bit Registered ALU | 3 |
| SN74ACT8836A 32- × 32-Bit Parallel Multiplier | 4 |
| SN74ACT8841A Digital Crossbar Switch | 5 |
| SN74ACT8847 64-Bit Floating-Point/Integer Unit | 6 |
| SN74ACT8847 Application Information | 7 |
| SN74ACT8867 32-Bit Vector Processor Unit | 8 |
| Design Support | 9 |
| Mechanical Data | 10 |

| | |
|----|---|
| 1 | General Information |
| 2 | EN7A0CT0B1BA 16-Bit Microcontroller |
| 3 | EN7A0CT0B32A 32-Bit Registered ALU |
| 4 | EN7A0CT0B58A 32 x 32-Bit Parallel Multiplier |
| 5 | EN7A0CT0B84A Digital Transceiver Switch |
| 6 | EN7A0CT0BNT 64-Bit Floating-Point Register File |
| 7 | EN7A0CT0B7A7 Application Information |
| 8 | EN7A0CT0B57 32-Bit VLSI Processor Unit |
| 9 | Design Support |
| 10 | Mechanical Data |

Design Support

Design Support for TI's SN74ACT8800 Family

TI's '8800 32-bit processor family is supported by a variety of tools developed to aid in design evaluation and verification. These tools will streamline all stages of the design process, from assessing the operation and performance of an individual device to evaluating a total system application. The tools include functional models, behavioral models, microcode development software, as well as the expertise of TI's Datapath VLSI Products Systems Engineering group.

Functional Evaluation Models Aid in Device Evaluation

Many design decisions can easily be made and evaluated before hardware or board prototypes are needed, using functional evaluation software models. The result is shortened design cycles and lower design costs.

Texas Instruments offers functional evaluation models for many of the devices in the '8800 family. These models are written in Microsoft C® and can be used in stand-alone mode or as callable functions.

These models are designed to provide insight into the operation of the devices by allowing the designer to write microcode and run it through the model. This allows the designer to select the device that best executes a specific application and provides a head start in evaluating programming performance.

The models correctly represent device timing in clock cycles, measured from the input of control and data to the output of results and status. Hence, initial performance estimates for a particular design can be made by relating the number of clock cycles required for an operation to the typical ac timing data for the device.

Behavioral Simulation Models Simplify System Debugging

System simulation with behavioral models can further shorten design time and ease design effort. The behavioral simulation models that support TI's '8800 chip set have the timing-control and error-handling capability to perform thorough PCB and system simulation. These models decrease the time spent in debugging and reduce the number of required prototype runs.

Users of system simulation models report a reduction by more than half in the number of prototype runs typically required to produce the highest-quality system. This savings in time reduces costs and gets the product to market as much as several months earlier than could be done using traditional methods.

Microsoft C is a registered trademark of Microsoft Corporation

Behavioral models for TI's '8800 family are written at the functional behavioral level and, therefore, are faster and easier to use and take up less disk space than some other types of simulation models. This higher efficiency means a simulation run can include more IC models and yet require less CPU time than an equivalent simulation using other types of models.

These behavioral simulation models also provide explicit error messages that can help in the debugging process. For example, if a design violates a device set-up time, the model explains, via an error message, what type of violation occurred, at what point it occurred in the simulation run, and specifically which part's set-up time was violated. Then, the model continues on with the run as if no violation occurred, saving time rather than crashing the run at every error.

In other words, an expert debugger is built right into the simulation.

The models are available with commercial and military timing and interact with a variety of simulators.

Behavioral Models for TI's '8800 Family are Easily Obtained

Texas Instruments has been working closely with both Quadtree Software Corporation and Logic Automation Incorporated to produce software behavioral simulation models of many of its VLSI devices. Since accuracy is key to solving design problems, we've provided Quadtree and Logic Automation with test patterns for most of our devices to ensure each model passes the same set of test vectors as does the actual silicon device.

Quadtree offers a library of Designer's Choice™ full-functional behavioral models of Texas Instruments '8800 32-bit processor building block devices.

Logic Automation Smartmodel™ library contains many Texas Instruments products, including devices from the '8800 chip set.

These companies may be contacted directly at the addresses below. General information about behavioral model support for the '8800 family may be obtained by calling Texas Instruments at (214) 997-5402.

LOGIC AUTOMATION INCORPORATED
P.O. Box 310
Beaverton, OR 97075
(503) 690-6900

QUADTREE SOFTWARE CORPORATION
1170 Route 22 East
Bridgewater, NJ 08807
(201) 725-2272

Quadtree and Designer's Choice are trademarks of Quadtree Software Corporation
Logic Automation and Smartmodel are trademarks of Logic Automation Incorporated

'8800 SDB Design Kit

TI offers an '8800 Software Development Board (SDB) Design Kit as an evaluation and training tool. The '8800 SDB kit uses a range of software development tools to allow users to evaluate performance and write microprograms for several of the '8800 building blocks. Using the SDB, microcode can be developed earlier in a system's design cycle so that code development parallels, rather than follows, prototype design.

The '8800 SDB Design Kit consists of a combination of specially developed hardware, software, and documentation including:

- The '8800 Software Development Board Assembly
- The '8800 SDB User's Guide
- Floppy disk with MS-DOS™ software tools written in Microsoft C, several example microprograms, and demo programs. Source code is included.
- Microcode definition files for use with HILEVEL, STEP Engineering, and Texas Instruments microcode development tools.

Built on a PC/AT card occupying a single slot, the '8800 SDB contains an 'ACT8818A microsequencer, 'ACT8832A registered ALU, and an 'ACT8847 floating-point/integer unit, along with 32 K by 128 bits of microcode memory, and 32 K by 32 bits of local data memory. A block diagram of the '8800 SDB is detailed in Figure 8-1. The board operates under an MS-DOS environment.

The SDB Design Kit complements other '8800 family development tools such as functional evaluation and behavioral simulation models. It actually provides the next step beyond simulators. System code can be executed in a realtime environment that includes conditional branching, on-board data memory, and single-step/breakpoint facilities.

For additional technical information, contact Datapath VLSI Products Systems Engineering at (214) 997-3970. For ordering information, please call your local field sales representative.

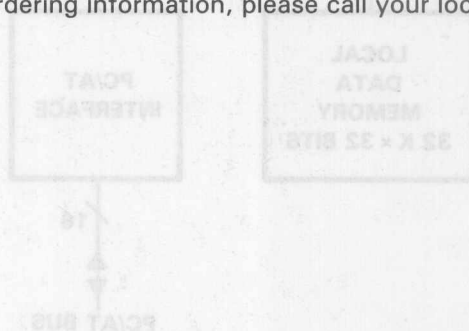


Figure 8-1: '8800 SDB Block Diagram

MetaStep is a trademark of STEP Engineering, Inc.

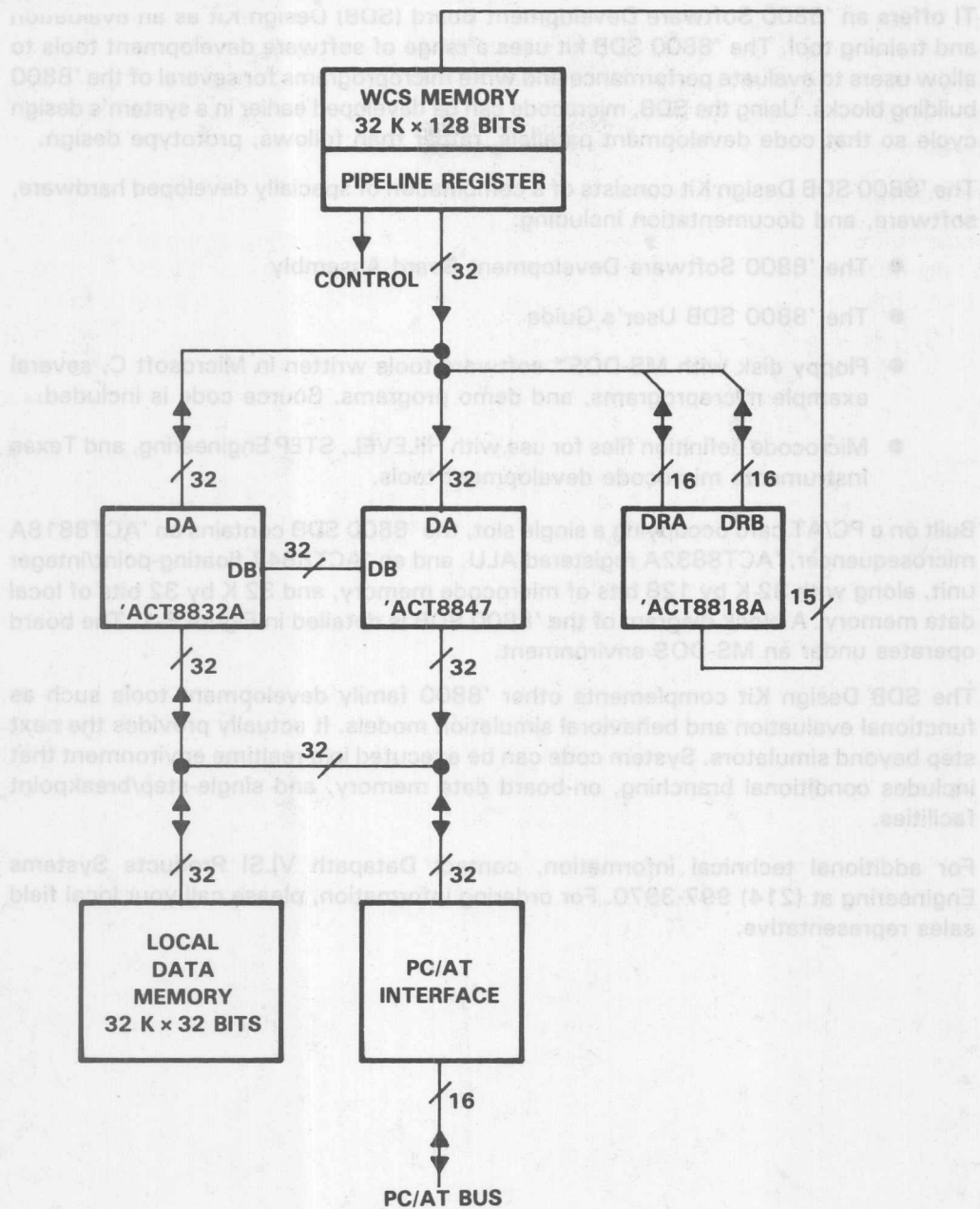


Figure 8-1. '8800 SDB Block Diagram

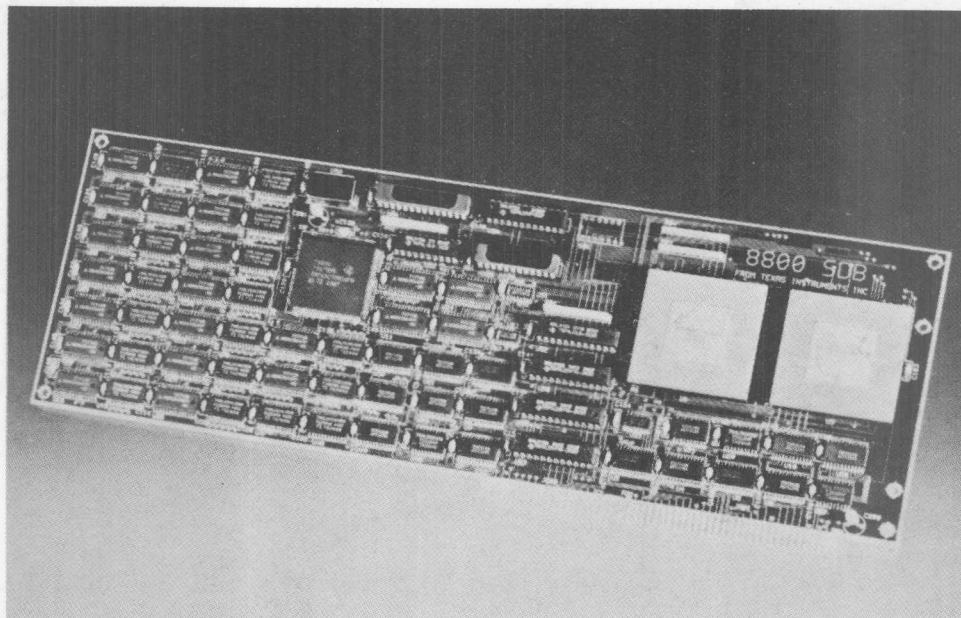
Program Code Generation Using the TI Meta Assembler

The TI Meta Assembler (TIM) provides the means to create object microcode files and to support listings for programs that execute in architectures without standard instruction sets. The end-product of TIM is an absolute object code module in suitable format for downloading to PROM programmers or to the emulator memories of development systems. TIM is fully compatible with some other assemblers as well.

Systems Expertise is a Phone Call Away

Texas Instruments Datapath VLSI Products Systems Engineering group is available to help designers analyze TI's high-performance VLSI products, such as the '8800 32-bit processor family. The group works directly with designers to provide ready answers to device-related questions and also prepares a variety of applications documentation.

The group may be reached in Dallas, at (214) 997-3970.



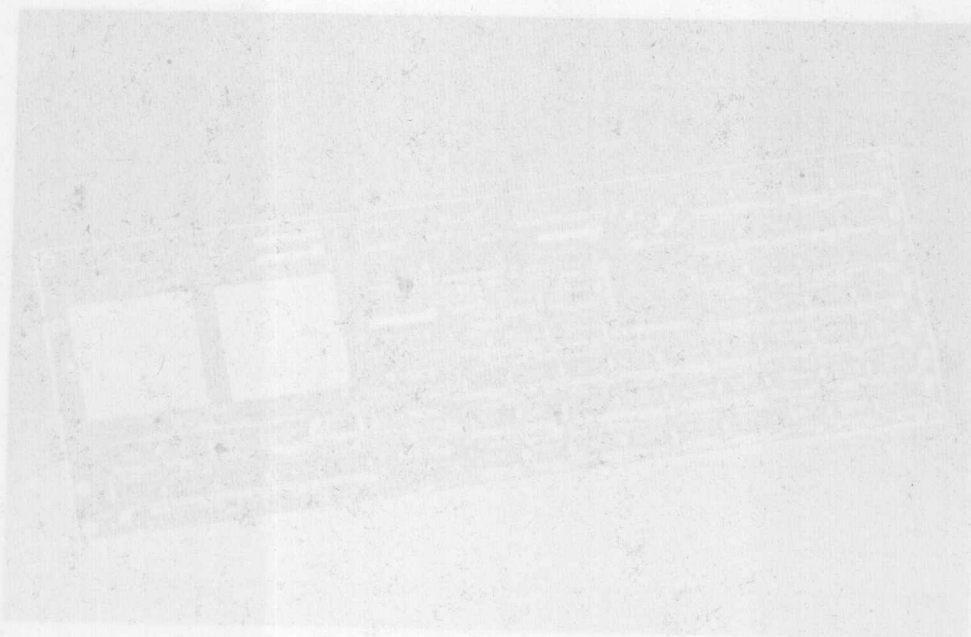
Program Code Generation Using the TI Meta Assembler

The TI Meta Assembler (TIM) provides the means to create object microcode files and to support listings for programs that execute in architectures without standard instruction sets. The end-product of TIM is an absolute object code module in suitable format for downloading to PROM programmers or to the emulator memories of development systems. TIM is fully compatible with some other assemblers as well.

Systems Expertise is a Phone Call Away

Texas Instruments Datapath VLSI Products Systems Engineering group is available to help designers analyze T's high-performance VLSI products, such as the 8800 32-bit processor family. The group works directly with designers to provide ready answers to device-related questions and also prepares a variety of applications documentation.

The group may be reached in Dallas, at (214) 987-3870.



General Information

1

SN74ACT8818A 16-Bit Microsequencer

2

SN74ACT8832A 32-Bit Registered ALU

3

SN74ACT8836A 32- × 32-Bit Parallel Multiplier

4

SN74ACT8841A Digital Crossbar Switch

5

SN74ACT8847 64-Bit Floating-Point/Integer Unit

6

SN74ACT8847 Application Information

7

SN74ACT8867 32-Bit Vector Processor Unit

8

Design Support

9

Mechanical Data

10

Mechanical Data

SN74ACT8818A 11 × 11 GC Package

SN74ACT8832A 17 × 17 GB Package

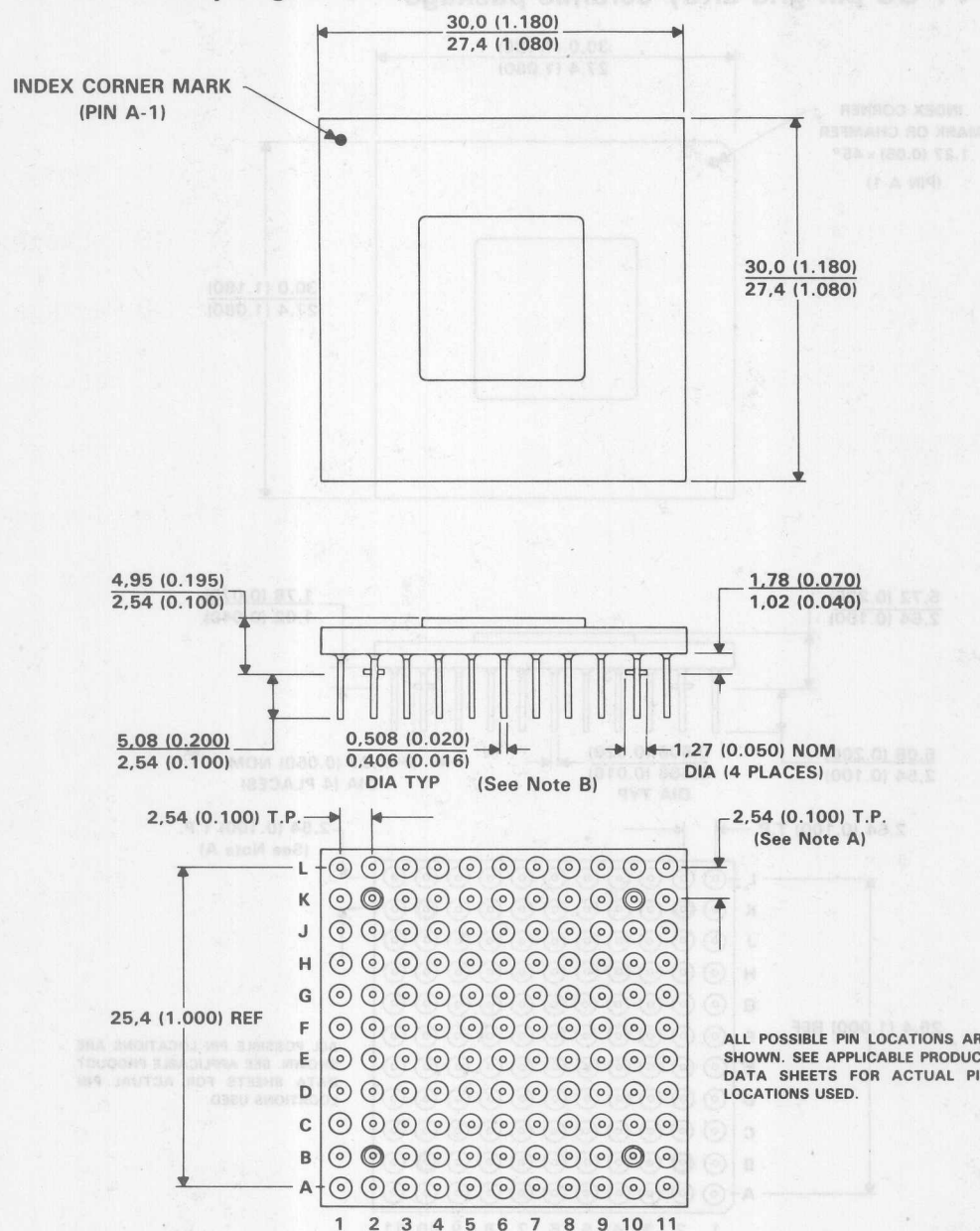
SN74ACT8836A 15 × 15 GB Package

SN74ACT8841A 15 × 15 GC Package

SN74ACT8847 17 × 17 GA Package

SN74ACT8867 17 × 17 GA Package

11 × 11 GB pin grid array ceramic package

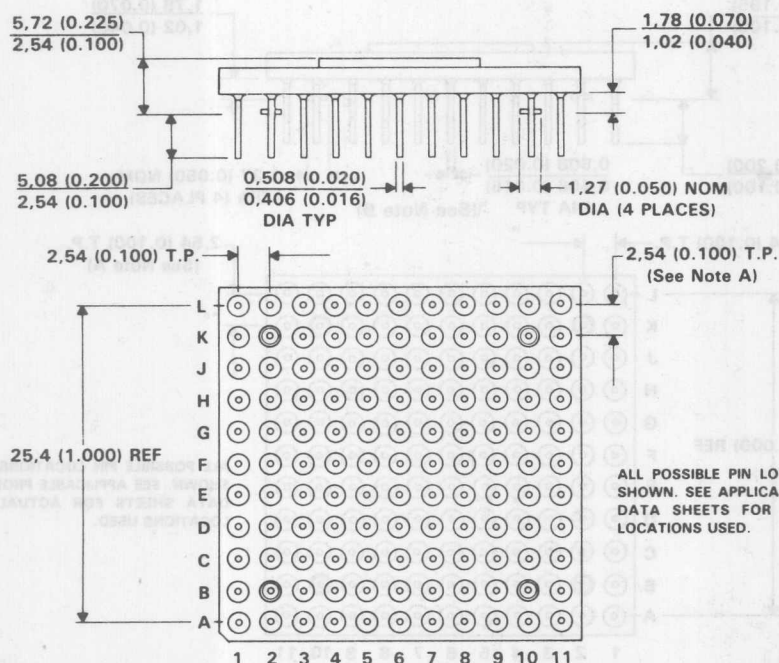
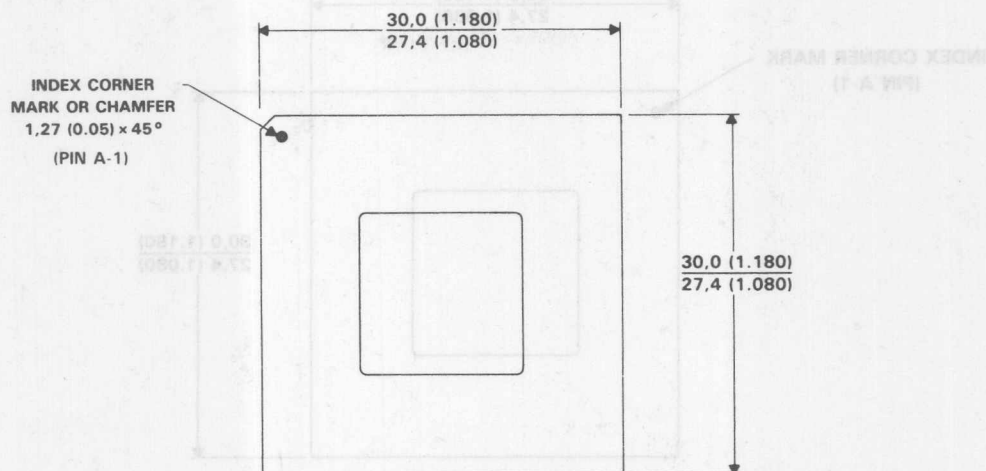


ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES

NOTES: A. Pin tips are located with 0,13 (0.005) radius of true position relative to each other at maximum material condition and within 0,46 (0.018) radius relative to the center of the ceramic.

B. Dimensions do not include solder finish.

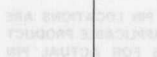
11 × 11 GC pin grid array ceramic package



ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES

NOTES: A. Pin tips are located with 0,13 (0.005) radius of true position relative to each other at maximum material condition and within 0,46 (0.018) radius relative to the center of the ceramic.

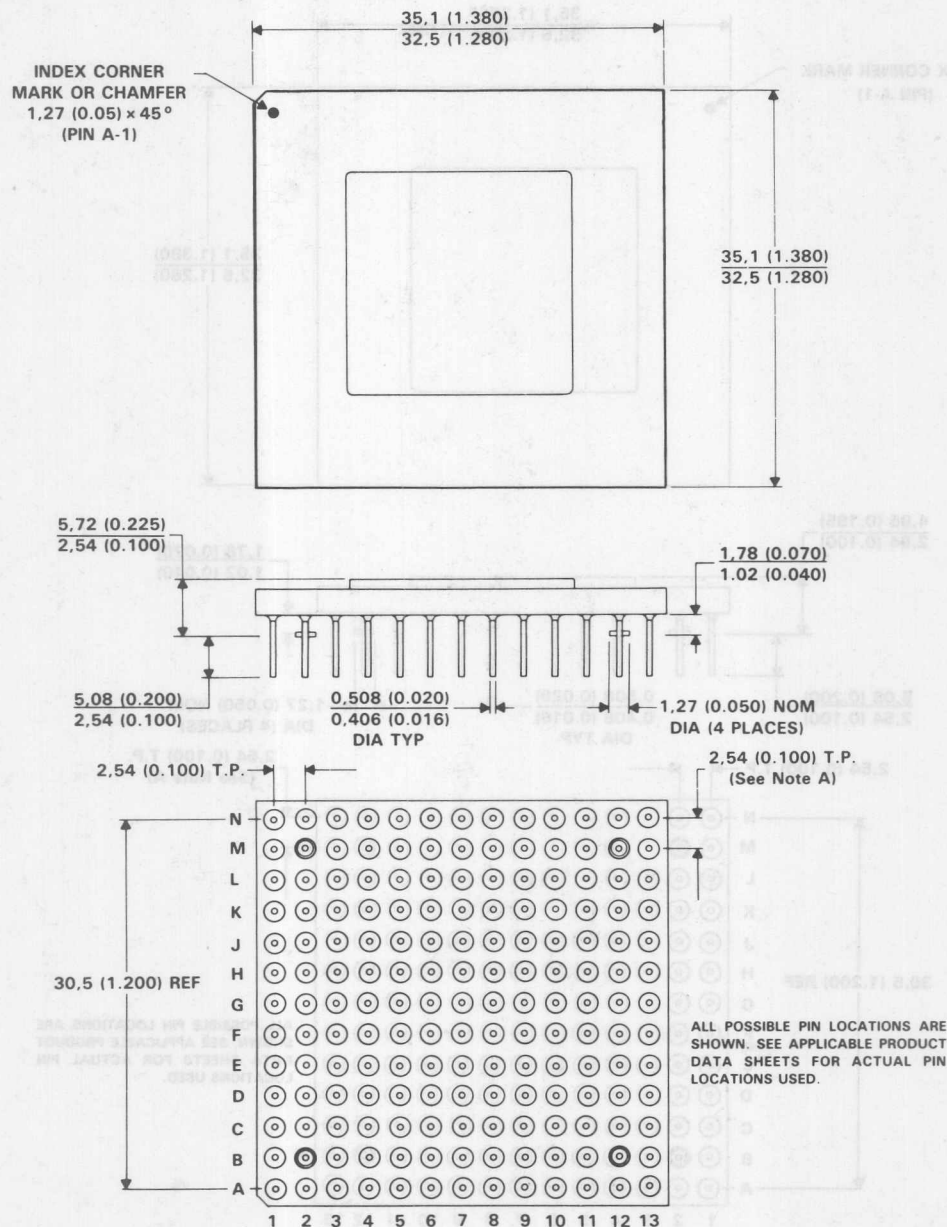
B. Dimensions do not include solder finish.



ALL POSSIBLE PIN LOCATIONS ARE SHOWN. SEE APPLICABLE PRODUCT DATA SHEETS FOR ACTUAL PIN LOCATIONS USED.

relative to the c

13 × 13 GC pin grid array ceramic package

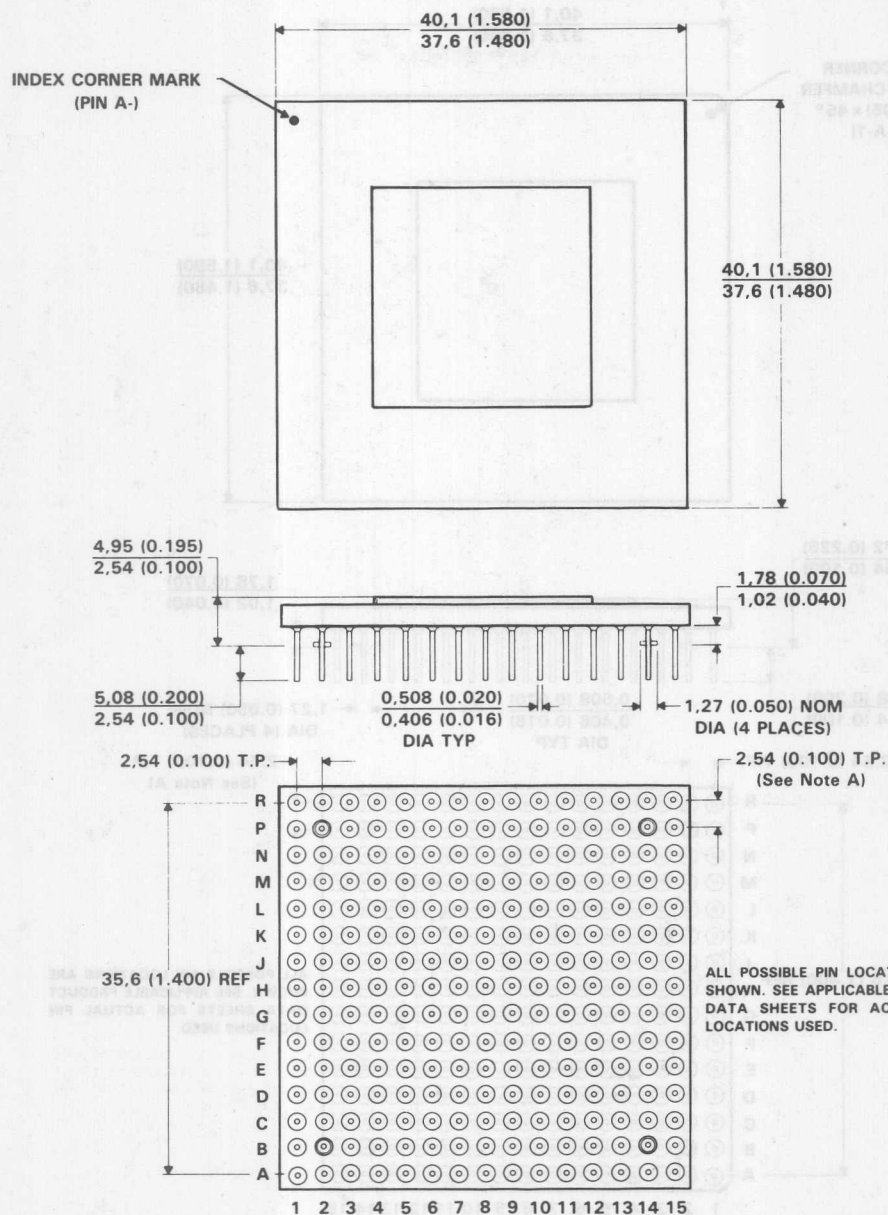


ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES

NOTES: A. Pin tips are located with 0,13 (0.005) radius of true position relative to each other at maximum material condition and within 0,46 (0.018) radius relative to the center of the ceramic.

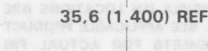
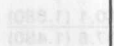
B. Dimensions do not include solder finish.

15 × 15 GB pin grid array ceramic package



ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES

- NOTES: A. Pin tips are located with 0,13 (0.005) radius of true position relative to each other at maximum material condition and within 0,46 (0.018) radius relative to the center of the ceramic.
- B. Dimensions do not include solder finish.



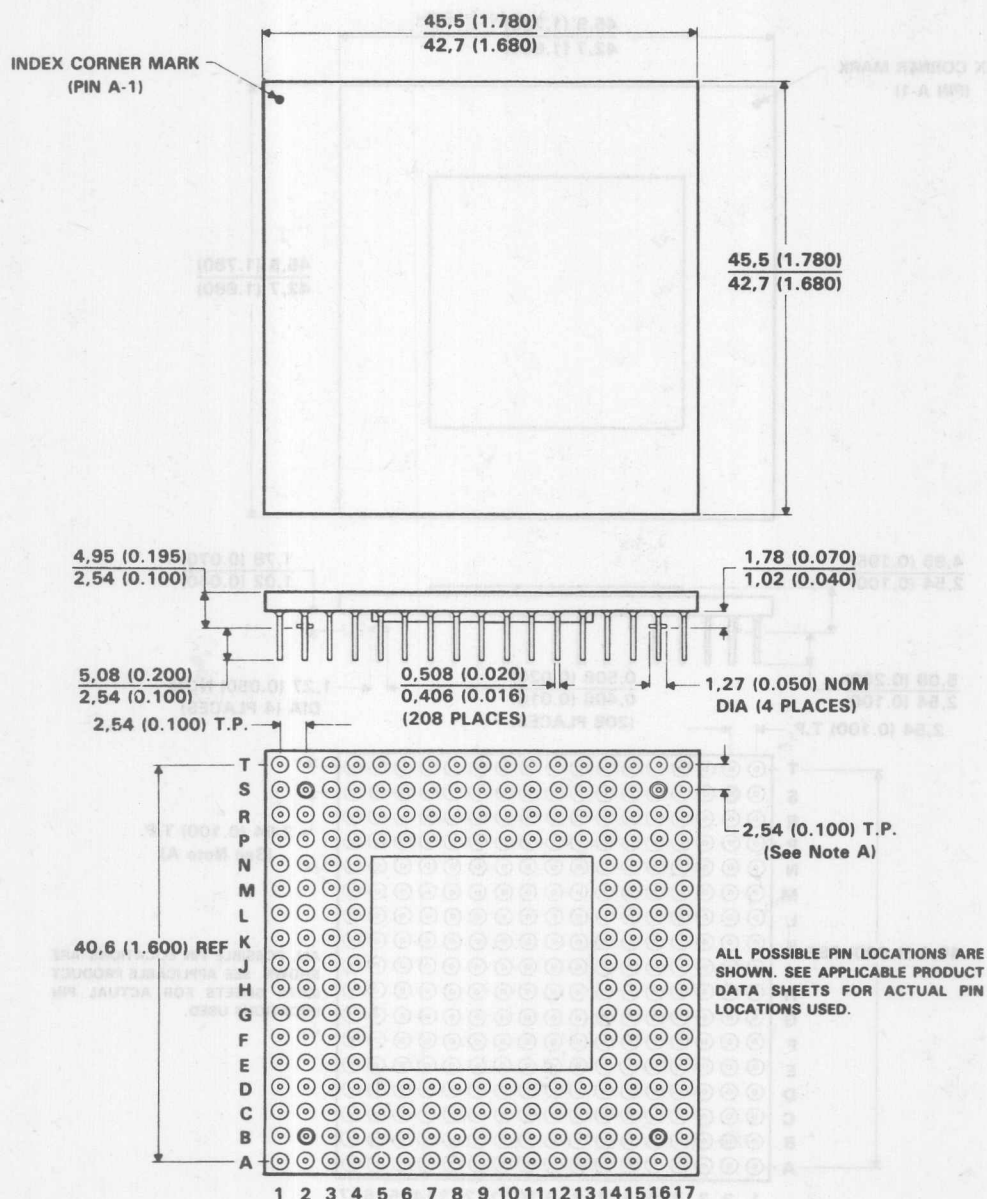
ALL POSSIBLE PIN LOCATIONS ARE SHOWN. SEE APPLICABLE PRODUCT DATA SHEETS FOR ACTUAL PIN LOCATIONS USED.

ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES

to attract

B. Dimensions do not include solder finish.

17 × 17 GA pin grid array ceramic package

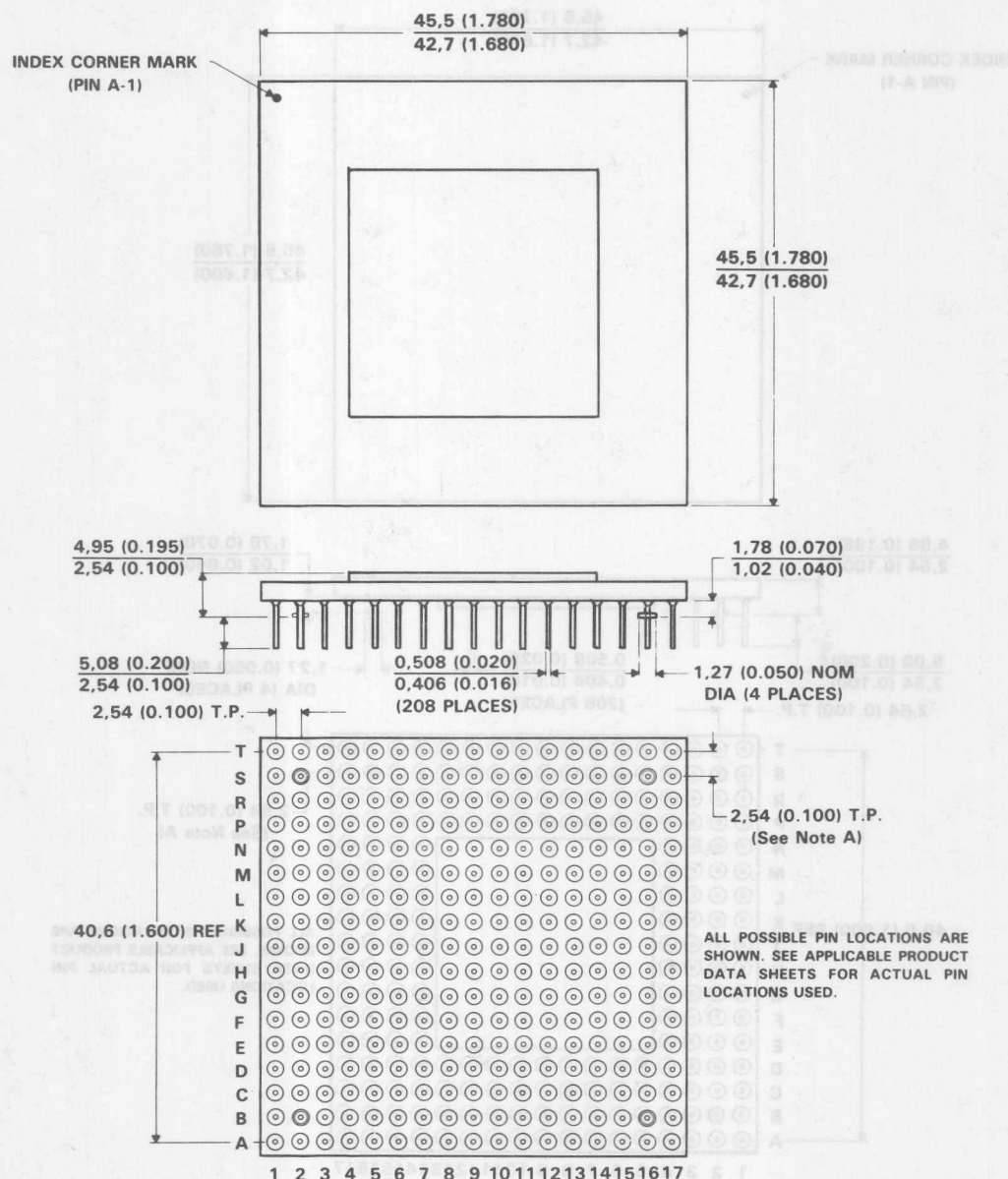


ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES

NOTES: A. Pin tips are located with 0,13 (0.005) radius of true position relative to each other at maximum material condition and within 0,46 (0.018) radius relative to the center of the ceramic.

B. Dimensions do not include solder finish.

17 × 17 GB pin grid array ceramic package



ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES

- NOTES: A. Pin tips are located with 0,13 (0.005) radius of true position relative to each other at maximum material condition and within 0,46 (0.018) radius relative to the center of the ceramic.
- B. Dimensions do not include solder finish.